Sparsity-Preserving Encodings for Straggler-Optimal Distributed Matrix Computations at the Edge

Anindya Bijoy Das, Aditya Ramamoorthy, David J. Love and Christopher G. Brinton

***Abstract*—Matrix computations are a fundamental building-block of edge computing systems, with a major recent uptick in demand due to their use in AI/ML training and inference procedures. Existing approaches for distributing matrix computations involve allocating coded combinations of submatrices to worker nodes, to build resilience to slower nodes, called stragglers. In the edge learning context, however, these approaches will compromise sparsity properties that are often present in the original matrices found at the edge server. In this study, we consider the challenge of augmenting such approaches to preserve input sparsity when distributing the task across edge devices, thereby retaining the associated computational efficiency enhancements. First, we find a lower bound on the weight of coding, i.e., the number of submatrices to be combined to obtain coded submatrices, to provide the resilience to the maximum possible number of straggler devices (for given number of devices and their storage constraints). Next we propose distributed matrix computation schemes which meet the exact lower bound on the weight of the coding. Numerical experiments conducted in Amazon Web Services (AWS) validate our assertions regarding straggler mitigation and computation speed for sparse matrices.**

***Index Terms*—Distributed computing, MDS Codes, Stragglers, Sparsity, IoT/edge heterogeneity**

I. Introduction

Edge computing platforms are constantly struggling to keep pace with the escalating demands for data processing tasks. A vast volume of data is being generated at the network edge today due to the rise of the Internet of Things (IoT), which encompasses self-driving cars, drones, health monitoring devices, and many other intelligent applications [2]. The mounting complexity of edge learning tasks, exemplified by the constantly growing sizes of deep neural networks and other AI/ML models, coupled with vast amounts of data used in training and inference, persistently obstruct scalability. As these models become more sophisticated, they necessitate increasingly robust computational resources and capabilities.

A possible solution to this can be borrowed from distributed computing systems, where computationally intensive tasks are distributed across multiple worker nodes. In the edge computing context, an oversubscribed edge server can allocate a processing task across multiple edge/IoT devices that has access to. Specifically, a profoundly-complex learning task can undergo partitioning into numerous sub-learning tasks, which are subsequently dispatched to multiple edge devices for execution [3]. Thus, the computational burden at the edge server can be parallelized, enhancing training speed.

In the edge learning context, particular attention must be paid to *matrix computations*, which form the cornerstone of numerous data processing tasks in AI/ML. With the expansion of data sizes, these computations encompass high-dimensional matrices, leading to extended runtimes with all else held constant. Distributed edge computations will aim to mitigate this by segmenting the entire matrix operation into smaller submatrices, and dispersing them across multiple edge devices for parallel execution. Nonetheless, the overall execution time of a task can be markedly impacted by slower or failed edge devices, often referred to as "stragglers" [4].

Stragglers are prevalent across the edge due to different reasons including heterogeneous computation capabilities found across devices [4], [5] and network congestion [6], [7], and can pose challenges to completing edge learning tasks in a timely manner. Since the system must wait for the slowest task to finish before proceeding, these are particularly problematic in environments where tasks are expected to complete synchronously. Scenarios affected by stragglers encompass various real-world applications in federated learning [8]–[10], blockchain [11]–[13], timely computing [14], [15], distributed optimization [16]–[19], smart healthcare [20], [21] and other similar environments. Thus, mitigating the stragglers is essential for optimizing the synchronization and speed of task completion in such cases.

Several coding theory techniques [22]–[33] have been recently proposed to mitigate the effect of stragglers. A simple example [22] for computing $\mathbf{A}^T\mathbf{x}$ using three devices involves partitioning the matrix $\mathbf{A}$ into two block-columns, denoted as $\mathbf{A} = [\mathbf{A}_0|\mathbf{A}_1]$. The edge devices are then assigned specific tasks: one computes $\mathbf{A}_0^T\mathbf{x}$, another computes $\mathbf{A}_1^T\mathbf{x}$, and the third computes $(\mathbf{A}_0 + \mathbf{A}_1)^T\mathbf{x}$. Each device then handles only half of the computational load, the system can recover $\mathbf{A}^T\mathbf{x}$ if any two out of the devices return their results. This implies that the system can withstand the failure or delay of a single straggler. In broader terms, the recovery threshold stands as a pivotal metric, denoting the minimum number of edge devices (τ) needed to fulfill their tasks, thereby facilitating the recovery of $\mathbf{A}^T\mathbf{x}$ from any subset of τ devices. Note that this idea can be extended to matrix-matrix multiplication also.

While numerous existing works achieve the optimal recovery threshold [23], [25], [32], [33] for specific device and storage

Anindya Bijoy Das, David J. Love, and Christopher. G. Brinton are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA 47907 (e-mail: {das207, djlove, cgb}@purdue.edu).

Aditya Ramamoorthy is with the Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, USA 50011 (email: aditya@iastate.edu).

The material in this work has appeared in part [1] at the 59th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Monticello, IL, USA, 2023, [DOI: 10.1109/Allerton58177.2023.10313473].

This work was supported in part by the National Science Foundation (NSF) under Grants CNS-2212565, ITE-2326898 and CCF-2115200, and Office of Naval Research (ONR) under grant N000142112472.

constraints, they are not exempt from limitations. Real-world datasets, pervasive across various fields like optimization, deep learning, power systems, and computational fluid dynamics, frequently manifest as sparse matrices. Leveraging this sparsity effectively holds the potential to significantly reduce the overall time required for matrix computations. However, techniques relying on MDS codes tend to generate dense linear combinations of submatrices [23], [25], [32], [33], thereby obliterating the inherent sparsity within the matrix structure. Consequently, this can result in a severe reduction in the computational speed of edge devices [34]. In addition, this can lead to further delay, since the central server needs to transmit a larger number of non-zero entries to the edge devices. In this paper, our primary objective is to devise methodologies that integrate a relatively modest number of submatrices while preserving an optimal recovery threshold. This can enhance the computation speed and reduce the transmission delay, thereby improving the overall job completion speed.

Organization: In this work, first we formulate the problem, provide a literature background and summarize our contributions in Sec. II. Then, in Sec. III, we find a lower bound on the number of submatrices to be combined for coded submatrices that will provide resilience to the maximum number of stragglers in a system with given storage and computation constraints. After that, we develop novel approaches for distributed matrix-vector multiplication (Sec. IV) and distributed matrix-matrix multiplication (Sec. V), both of which meet that lower bound, maximizing sparsity preservation while providing resilience to the maximum number of stragglers. Next, we carry out experiments on an Amazon Web Services (AWS) cluster, and provide the numerical results in Sec. VI. Finally, Sec. VII concludes the paper with several possible future directions.

II. Problem Formulation, Background and Summary of Contributions

In this section, we initially outline the problem of addressing sparsity in distributed matrix computations. Following that, we provide a concise overview of existing methods, analyze their limitations, and outline the key contributions of our research.

A. Problem Formulation

In this work, we investigate a distributed computing system comprised of a edge server and a set of n edge devices. The goal is to compute $\mathbf{A}^T\mathbf{x}$ for matrix-vector multiplication or $\mathbf{A}^T\mathbf{B}$ for matrix-matrix multiplication. Here, $\mathbf{A} \in \mathbb{R}^{t\times r}$, $\mathbf{B} \in \mathbb{R}^{t\times w}$ are the "input" matrices, and $\mathbf{x} \in \mathbb{R}^t$ is a vector.

First, we consider a coded matrix-vector multiplication scheme where The primary objective of this system is to calculate the product $\mathbf{A}^T\mathbf{x}$, where $\mathbf{A}$ represents a sparse matrix and $\mathbf{x}$ denotes a vector. In line with previous approaches, we initially partition matrix $\mathbf{A}$ into k_A distinct block-columns, $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{k_A-1}$; hence the goal is to recover k_A corresponding unknowns $\mathbf{A}_0^T\mathbf{x}, \mathbf{A}_1^T\mathbf{x}, \mathbf{A}_2^T\mathbf{x}, \ldots, \mathbf{A}_{k_A-1}^T\mathbf{x}$.

Next, we consider the matrix-matrix multiplication case, where matrices $\mathbf{A}$ and $\mathbf{B}$ are partitioned into k_A and k_B disjoint block-columns, as $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{k_A-1}$ and $\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_{k_B-1}$. Thus, in this case we need to recover, in total, k_Ak_B unknowns in the form of $\mathbf{A}_i^T\mathbf{B}_j$ where $0 \le i \le k_A - 1$ and $0 \le j \le k_B - 1$.

The assumption is that the edge devices possess uniform memory capacity and computational speed. To elaborate, each worker can retain $\gamma_A = \frac{1}{k_A}$ portion of matrix $\mathbf{A}$, as well as the complete vector $\mathbf{x}$ (in the matrix-vector case), or $\gamma_B = \frac{1}{k_B}$ portion of matrix $\mathbf{B}$ (in the matrix-matrix case). In real-world scenarios, stragglers might emerge owing to discrepancies in computational speeds or instances of failure among designated edge devices at particular times [23].

In this work, we allocate to each edge device a randomized linear combination of specific block-columns from matrix $\mathbf{A}$, along with the vector $\mathbf{x}$ in the matrix-vector case. For matrix-matrix multiplication, each edge device receives a randomized linear combination of certain block-columns from $\mathbf{A}$ and another combination from $\mathbf{B}$. However, as discussed in Section I, assigning dense linear combinations risks losing the inherent sparsity of the matrices involved. To circumvent this, our aim is to distribute linear combinations involving fewer submatrices [29], [31]. To quantify this strategy, we introduce the notion of "weight", which plays a pivotal role in handling sparse matrices in distributed matrix computations.

Definition 1. We define the "weight" (ω) for a coded matrix-computation scheme as the number of unknowns that participate within any matrix product computed by each of the edge device. Thus, if we combine ω_A submatrices of $\mathbf{A}$ to obtain the encoded submatrices for matrix-vector multiplication, we have $\omega = \omega_A$. In the matrix-matrix case, if we combine ω_A and ω_B submatrices of $\mathbf{A}$ and $\mathbf{B}$, respectively, to obtain the assigned encoded submatrices, then $\omega = \omega_A\omega_B$.

Thus, our goal is to obtain the optimal recovery threshold ($\tau = k_A$ for the matrix-vector case, and $\tau = k_Ak_B$ for the matrix-matrix case [25]) while maintaining ω as low as possible.

B. Existing Methods and Our Motivations

Numerous coded computation schemes have been recently proposed for distributed matrix computations [22]–[25], [27]–[30], [32], [33], [35]–[41]. In this section, we provide an overview of existing algorithms and examine their limitations, which have inspired our current research efforts. Note that there are methods in [35], [40]–[42] which are developed for matrix-vector multiplication only. In this work, in addition to the matrix-vector case, we also address the distributed matrix-matrix multiplication scenario, the more challenging one.

The initial studies in this domain focused on the recovery threshold as a pivotal metric. Unlike the methodologies outlined in [34], [40], which exhibit suboptimal straggler resilience, the polynomial code approach emerges as one of the pioneering methods to achieve the optimal recovery threshold. We begin by illustrating a simplified example of this approach [25] in the context of distributed matrix-matrix multiplication.

Let us consider a system consisting of $n = 5$ edge devices, each capable of storing half of each of the matrices $\mathbf{A}$ and $\mathbf{B}$, hence $1/k_A = 1/k_B = 1/2$. We partition $\mathbf{A}$ and $\mathbf{B}$ into $k_A = k_B = 2$ block-columns each, denoted as $\mathbf{A}_0$, $\mathbf{A}_1$ and $\mathbf{B}_0$, $\mathbf{B}_1$, respectively. Defining matrix polynomials $\mathbf{A}(z) =$

$\mathbf{A}_0 + \mathbf{A}_1 z$ and $\mathbf{B}(z) = \mathbf{B}_0 + \mathbf{B}_1 z^2$, we can write the product of these two matrix polynomials as $\mathbf{A}^T(z)\mathbf{B}(z) = \mathbf{A}_0^T\mathbf{B}_0 + \mathbf{A}_1^T\mathbf{B}_0 z + \mathbf{A}_0^T\mathbf{B}_1 z^2 + \mathbf{A}_1^T\mathbf{B}_1 z^3$. The edge server assesses $\mathbf{A}(z)$ and $\mathbf{B}(z)$ at $n = 5$ distinct real numbers, and then, transmits the corresponding evaluated matrices to edge device W_i, where $0 \leq i \leq n-1$. Each device then computes its designated matrix-matrix block-product and sends back the result to the server. Given that $\mathbf{A}^T(z)\mathbf{B}(z)$ forms a degree-3 polynomial, upon receiving results from the fastest $\tau = 4$ devices, the server can decode all coefficients within $\mathbf{A}^T(z)\mathbf{B}(z)$, thereby obtaining $\mathbf{A}^T\mathbf{B}$ (since any 4×4 submatrix of a 5×4 Vandermonde matrix has a rank 4). Hence, the recovery threshold is $\tau = 4$, indicating resilience to $s = 1$ straggler.

For given storage constraints, i.e., storing $1/k_A$ and $1/k_B$ fractions of $\mathbf{A}$ and $\mathbf{B}$, respectively, if each device is responsible to compute $1/k$ fraction of the overall job (where $k = k_A k_B$), then $s = n - k_A k_B$ is the maximum number of stragglers that any scheme can be resilient to [25]. This can be an important property of a coded computation scheme, since resilience to a high number of stragglers is often crucial. For example, consider smart city IoT environments, where distributed learning is often incorporated within edge computing systems to provide intelligent, data-driven services [43], [44]. Quality of Service (QoS) for these smart city use-cases have been shown to greatly improve through computation offloading, and thus highly depends on the system being resilient to slower devices.

While the polynomial code meets the lower bound on the recovery threshold, recent works on matrix computations have identified metrics beyond recovery threshold that also need to be considered. Table I demonstrates an overall summary to compare available schemes in terms of different other metrics. Here we discuss the importance of factoring them into our methodology which aims at improving those other metrics along with enjoying the optimal recovery threshold.

Sparse Matrices: Sparse matrices are ubiquitous across various domains like optimization, deep learning, and electromagnetism, as evidenced by their prevalence in real-world datasets (see [49] for examples). Essentially, many practical scenarios involve matrices with sparse elements, offering a potential opportunity to reduce the edge device computation time significantly [34]. For example, for anomalous defect elimination in intelligent manufacturing, sparsity of the normal features can be very beneficial [50], [51]. This can significantly enhance the speed of the overall training process, which often involves deep neural networks [52], [53].

Consider two column vectors, $\mathbf{a}$ and $\mathbf{y}$, both of length m, where $\mathbf{a}$ contains roughly ψm non-zero entries ($0 < \psi << 1$). Computing $\mathbf{a}^T\mathbf{y}$ consumes about $2\psi m$ floating-point operations (FLOPs), in contrast to approximately $2m$ FLOPs which would be required for a dense vector $\mathbf{a}$. In a similar manner, the dense linear encoding strategies in [23], [25], [32], [33] inflate the number of non-zero entries in encoded matrices, thus forfeit sparsity preservation. For a system with storage coefficients $1/k_A$ and $1/k_B$, the polynomial coding scheme [25] and its derivatives [32] obtain the encoded submatrices of $\mathbf{A}$ and $\mathbf{B}$ by having linear combinations of k_A and k_B submatrices, respectively. Consequently, non-zero entries can increase by up to k_A and k_B times, respectively, compared to the original matrices, substantially elongating computation time. This underscores the necessity for schemes that minimize the fusion of uncoded submatrices.

Note that there are various methods in the literature [29], [34]–[36], [54] that highlight and leverage the inherent sparsity of the "input" matrices. However, they have other limitations. For example, the approach detailed in [35] is only applicable to the matrix-vector case; does not apply to matrix-matrix multiplication. The assumptions in [54] differ from ours as they involve the edge server in some matrix computations. Furthermore, the approach in [34] and the β-level coding scheme in [36] do not meet the optimal recovery threshold, necessitating more devices to complete their tasks. On the other hand, the straggler optimal approaches in [29], [36] assign multiple tasks to each edge device, some of which can be quite densely coded, leading to higher computation delay. The approach in [31] also addresses the sparsity issue, but it does not make any guarantees in terms of a theoretical lower bound on the encoding weights. This implies that an enhanced coding approach could further streamline computational speed beyond these techniques for sparse matrices. We provide theoretical guarantees for our proposed coding method in Sec. III.

Numerical Stability: The numerical stability of the system stands as another crucial concern. Given that the encoding and decoding techniques in coded computation function within the real field, decoding the unknowns from a system of equations may lead to highly inaccurate results if the corresponding system matrix is ill-conditioned. Round-off errors could magnify in the decoded outcome due to the elevated condition numbers of the decoding matrices. For instance, the polynomial code method outlined in [25] integrates Vandermonde matrices into the encoding process, known for their ill-conditioned nature. Literature addressing this challenge [23], [32], [33] underscores the significance of minimizing the worst-case condition number (κ_{worst}) across different choices of stragglers.

However, several numerically stable methods rely on random codes [23], [29], [33], [36], necessitating substantial time investment to find an optimal set of random coefficients to ensure the numerical stability of the system. This typically involves generating a set of random coefficients initially and assessing κ_{worst} across all straggler permutations. This step iterates several times (e.g., 20), retaining the set of coefficients yielding the minimum κ_{worst}. However, the latency introduced by this iterative process escalates with the number of edge devices, potentially causing delays in the encoding process. For example, the sparsely coded approaches in [29], [36] involve significant delay for determining a "good" set of coefficients, as demonstrated numerically in Sec. VI.

C. Summary of Contributions

- We address the straggler issue in distributed matrix computations for edge learning tasks, specifically focusing on sparse matrices. We define the concept of "weight" and determine its lower bound to ensure resilience against the

TABLE I: Comparison among existing works on coded matrix-computations (the approach in [45] involves a higher computational complexity). However, the methods in [35], [40]–[42] are developed for matrix-vector multiplication only, and therefore, are not included in the comparison. Note that the method in [31] does not develop and meet the lower bound on the encoding weight, which is one of our contributions.

CODES	OPTIMAL THRESHOLD?	NUMERICAL STABILITY?	SPARSELY CODED?	HETEROGENEOUS SYSTEM?
REPETITION CODES	✗	✓	✓	✓
PRODUCT CODES [46], FACTORED CODES [30]	✗	✓	✗	✗
POLYNOMIAL CODES [25]	✓	✗	✗	✓
MATDOT CODES [45]	✓	✗	✗	✓
ORTHOGONAL POLYNOMIAL [32], RKRP CODE [33],	✓	✓	✗	✓
BIVARIATE POLYNOMIAL CODE [47]	✓	✗	✗	✓
CONVOLUTIONAL CODE [23], CIRCULAR ROT. MATRIX [48]	✓	✓	✗	✓
β-LEVEL CODING [36]	✗	✓	✓	✗
SPARSELY CODED STRAGGLER OPTIMAL SCHEME [36]	✓	✓	✓	✗
CLASS-BASED SCHEME [29]	✓	✓	✓	✗
CYCLIC CODE (WITH RANDOM COEFFICIENTS) [31]	✓	✓	✓	✓
Proposed Scheme	✓	✓	✓	✓

maximum number of stragglers, maximizing the system's tolerance to delays or inefficiencies.

- Then, we develop an algorithm (Alg. 1) for distributed matrix-vector multiplication, which meets the exact lower bound to build resilience against maximum number of stragglers for any number of edge devices (n) and for any given storage constraint.
- Next, we develop an algorithm (Alg. 2) for straggler-resilient distributed matrix-matrix multiplication, which also meets the lower bound on the weight for different values of n, k_A and k_B. Both of Alg. 1 and Alg. 2 can be extended to a heterogeneous setting where the edge devices can have different storage and computation abilities.
- Subsequently, we analyze the per edge device computational complexity for our algorithms, and show that our approaches involve lower computational complexity than other recent approaches [29], [31]. In addition, we show that our approach also outperforms the approaches in [29], [36] requiring less time to find the encoding coefficients for the numerical stability of the system.
- Finally, we conduct extensive numerical experiments in the Amazon Web Services (AWS) cluster. Our results confirm the superiority of our approach to different dense coded approaches [25], [32], [33], and to the recent approaches developed specifically for sparse matrices [29], [31].

Note that a preliminary version of this paper appeared in [1]. Compared to the conference version, we have (i) developed Alg. 2 for the more challenging case: matrix-matrix multiplication, (ii) proved necessary theorems for straggler-resilience, and (iii) conducted the corresponding numerical simulations.

III. MINIMUM WEIGHT OF ENCODING

We assume homogeneous weights, i.e., the same number (ω_A) of uncoded $\mathbf{A}$ (and ω_B for $\mathbf{B}$) submatrices are combined to obtain the encoded $\mathbf{A}$ (and $\mathbf{B}$) submatrices. If we define ω as the number of participating unknowns in any computed results obtained from any edge device, in matrix-vector multiplication, we have $\omega = \omega_A$ and in the matrix-matrix case, $\omega = \omega_A\omega_B$. Now we state the following proposition which provides a lower bound on ω for any coded matrix computation scheme with resilience to $s = n - k$ stragglers, where $k = k_A$ for the matrix-vector case and $k = k_Ak_B$ for the matrix-matrix case.

Proposition 1. Consider a distributed computation system of n total devices each of which can store $1/k_A$ fraction of matrix $\mathbf{A}$ (and $1/k_B$ fraction of matrix $\mathbf{B}$ for the matrix-matrix multiplication case). Now, assume that a coded matrix computation scheme aims at resilience to $s = n - k$ stragglers out of n devices, where $k = k_A$ for the matrix-vector case and $k = k_Ak_B$ for the matrix-matrix case. Any such scheme that partitions $\mathbf{A}$ into k_A (and $\mathbf{B}$ into k_B) disjoint block-columns has to maintain a minimum homogeneous value for ω, which is given by $\hat{\omega} = \lceil \frac{(n-s)(s+1)}{n} \rceil$.

Proof. Since the scheme aims at resilience to *any* s stragglers, any scheme needs to ensure the presence of each of the corresponding k unknowns in at least $s+1$ different devices. In other words, for the matrix-vector case, each of $k = k_A$ such $\mathbf{A}_i$'s (and for the matrix-matrix case, each of $k = k_Ak_B$ such $\mathbf{A}_i^T\mathbf{B}_j$'s) has to participate within the encoded submatrices in at least $s + 1$ different devices. Thus, the sum of the required number of appearances of all the uncoded unknowns is $k(s+1)$.

Now, we assume homogeneous ω, i.e., each of these n devices is assigned a linear combination of ω uncoded unknowns. Hence, the total number of appearances of all the uncoded unknowns over the set of the worker nodes is $n\omega$.

Therefore, to build the resilience to *any* s stragglers, we need to have, $n\,\omega \geq k(s + 1)$, hence, $\omega \geq \frac{(n-s)(s+1)}{n}$.Thus, the minimum weight $\hat{\omega} = \left\lceil \frac{(n-s)(s+1)}{n} \right\rceil$. ■

Now we state a corollary which considers different values of k_A in terms of s, and provides the corresponding optimal weights for coded sparse matrix-vector multiplication.

Corollary 1. Consider the same setting as Prop. 1 for coded matrix computations. Now, we have the following cases.

- (i) If $k > s^2$, then $\hat{\omega} = s + 1$.
- (ii) If $s \leq k \leq s^2$, then $\lceil \frac{s+1}{2} \rceil \leq \hat{\omega} \leq s$.

Proof. Since $n = k + s$, from Prop. 1, we have

$$\hat{\omega} = \left\lceil \frac{k(s + 1)}{k + s} \right\rceil = \left\lceil \frac{1 + s}{1 + \frac{s}{k}} \right\rceil; \tag{1}$$

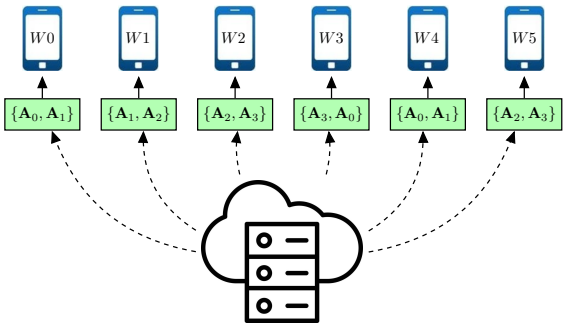

Fig. 1: Submatrix allocation by the edge server to a system with $n = 6$ devices, $s = 2$ stragglers and $\gamma_A = \frac{1}{4}$ according to Alg. 1. Here, the weight of every coded submatrix is $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil = 2$. Any $\{\mathbf{A}_i, \mathbf{A}_j\}$ indicates a random linear combination of $\mathbf{A}_i$ and $\mathbf{A}_j$.

hence, $\hat{\omega}$ is a non-decreasing function of k for fixed s.
Part (i): When $k > s^2$, we have $\frac{s}{k} < \frac{1}{s}$, and $\frac{1+s}{1+\frac{s}{k_A}} > \frac{1+s}{1+\frac{1}{s}} = s$. Thus, from (1), $\hat{\omega} > s$. In addition, from (1), for any $s \geq 0$, we have $\hat{\omega} \leq s + 1$. Thus, we have $\hat{\omega} = s + 1$.
Part (ii): If $k = s^2$, from (1), we have $\hat{\omega} = s$. Similarly, if $k = s$, from (1), we have $\hat{\omega} = \lceil \frac{s+1}{2} \rceil$. Thus, the non-decreasing property of $\hat{\omega}$ in terms of k concludes the proof. ■

Now we describe a motivating example below where the encoding scheme meets the lower bound mentioned in Prop. 1.

Example 1. Consider a toy system for distributed matrix-vector multiplication with $n = 6$ edge devices each of which can store $1/4$ fraction of matrix $\mathbf{A}$. We partition matrix $\mathbf{A}$ into $k_A = 4$ disjoint block-columns, $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3$. According to Prop. 1, the optimal weight $\omega = \omega_A$ can be as low as $\left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil = 2$. Now, we observe that the way the jobs are assigned in Fig. 1 meets that lower bound, where random linear combinations of $\omega_A = 2$ submatrices are assigned to the devices. It can be verified that this system has a recovery threshold $\tau = k_A = 4$, and thus, it is resilient to any $s = 2$ stragglers.

IV. Proposed Matrix-vector Multiplication Approach

In this section, we detail our overall approach for distributed matrix-vector multiplication which is outlined in Alg. 1. We partition matrix $\mathbf{A}$ into k_A block columns, $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{k_A-1}$, and assign a random linear combination of ω_A (weight) submatrices of $\mathbf{A}$ to every edge device. We show that for given n and k_A, our proposed approach provides resilience to maximum number of stragglers, $s = n - k_A$. In addition, our coding scheme maintains the minimum weight of coding as mentioned in Prop. 1.

Formally, we set $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil$, and assign a linear combination of $\mathbf{A}_i, \mathbf{A}_{i+1}, \ldots, \mathbf{A}_{i+\omega_A-1}$ (indices modulo k_A) to edge device W_i, for $i = 0, 1, \ldots, k_A - 1$, where the linear coefficients are chosen randomly from a continuous distribution. Next, we assign a random linear combination of $\mathbf{A}_{i\omega_A}, \mathbf{A}_{i\omega_A+1}, \mathbf{A}_{i\omega_A+2}, \ldots, \mathbf{A}_{(i+1)\omega_A-1}$ (indices modulo k_A) to edge device W_i, for $i = k_A, k_A + 1, \ldots, n - 1$. Note that every edge device also receives the vector $\mathbf{x}$. Once the fastest $\tau = k_A$ edge devices finish and return their computation results, the edge server decodes $\mathbf{A}^T\mathbf{x}$. Note that we assume $k_A \geq s$, i.e., at most *half* of the devices may be stragglers.

Algorithm 1: Proposed scheme for distributed matrix-vector multiplication

Input : Matrix $\mathbf{A}$, vector $\mathbf{x}$, n-number of edge devices, s-number of stragglers, storage fraction $\gamma_A = \frac{1}{k_A}$, such that $k_A \geq s$.

Partition $\mathbf{A}$ into k_A disjoint block-columns;
Set weight $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil$;
for $i \leftarrow 0$ **to** $n - 1$ **do**
  if $i < k_A$ **then**
    Define $T = \{i, i+1, \ldots, i+\omega_A-1\}$ (reduced modulo k_A);
  else
    Define $T = \{i\omega_A, i\omega_A+1, \ldots, (i+1)\omega_A-1\}$ (reduced modulo k_A);
  end
  Create a random vector $\mathbf{r}$ of length k_A with entries r_m, $0 \leq m \leq k_A - 1$;
  Create a random linear combination of $\mathbf{A}_q$'s where $q \in T$, thus $\tilde{\mathbf{A}}_i = \sum_{q \in T} r_q \mathbf{A}_q$;
  Assign encoded submatrix $\tilde{\mathbf{A}}_i$ and the vector $\mathbf{x}$ to edge device W_i;
  Edge device W_i computes $\tilde{\mathbf{A}}_i^T \mathbf{x}$;
end

Output : The edge server recovers $\mathbf{A}^T\mathbf{x}$ from the returned results by the fastest k_A devices.

A. Straggler Resilience Guarantee

Next we state the Lemma 1 which assists us to prove Theorem 1 that discusses straggler resilience of our scheme.

Lemma 1. Choose any $m \leq k_A$ edge devices out of all n devices in the distributed system. Now, if we assign the jobs to the edge devices according to Alg. 1, the total number of participating uncoded $\mathbf{A}$ submatrices within those m edge devices is lower bounded by m.

Proof. First we partition all n edge devices into *two* sets where the first set, $\mathcal{W}_0$ includes the first k_A devices and the second set, $\mathcal{W}_1$, includes the next s edge devices, i.e., we have

$$\mathcal{W}_0 = \{W_0, W_1, W_2, \ldots, W_{k_A-1}\}; \quad \text{and} \quad \mathcal{W}_1 = \{W_{k_A}, W_{k_A+1}, \ldots, W_{n-1}\}. \tag{2}$$

Thus, we have $|\mathcal{W}_0| = k_A$ and $|\mathcal{W}_1| = s \leq k_A$. Now, we choose any $m \leq k_A$ edge devices, where we choose m_0 devices from $\mathcal{W}_0$ and m_1 devices from $\mathcal{W}_1$, so that $m = m_0 + m_1$. We denote set of the participating uncoded $\mathbf{A}$ submatrices within those devices as $\mathcal{A}_0$ and $\mathcal{A}_1$, respectively. Hence, to prove the lemma, we need to show $|\mathcal{A}_0 \cup \mathcal{A}_1| \geq m$, for any $m \leq k_A$.

First, according to Alg. 1, we assign a random linear combination of $\mathbf{A}_i, \mathbf{A}_{i+1}, \mathbf{A}_{i+2}, \ldots, \mathbf{A}_{i+\omega_A-1}$ (indices modulo k_A) to edge device $W_i \in \mathcal{W}_0$. Thus, the participating submatrices

are assigned in a cyclic fashion [36], and the total number of participating submatrices within any m_0 devices of $\mathcal{W}_0$ is

$$|\mathcal{A}_0| \geq \min(m_0 + \omega_A - 1, k_A). \tag{3}$$

Next, we state the following claim for the number of participating submatrices in $\mathcal{W}_1$; the detailed proof is in [1].

Claim 1. Choose any $m_1 \geq \omega_A$ devices from $\mathcal{W}_1$. The number of participating submatrices within these devices, $|\mathcal{A}_1| = k_A$.

Case 1: If $m_1 \leq \omega_A - 1$, from (3) we have

$$\begin{aligned}|\mathcal{A}_0 \cup \mathcal{A}_1| \geq |\mathcal{A}_0| &= \min(m_0 + \omega_A - 1, k_A) \ \ (\text{when } m_0 > 0) \\ &\geq \min(m_0 + m_1, k_A) \geq m,\end{aligned}$$

since $m = m_0 + m_1 \leq k_A$. We take account the remaining scenario when $m_0 = 0$. In that case,

$$|\mathcal{A}_0 \cup \mathcal{A}_1| \geq |\mathcal{A}_1| \geq \omega_A \geq m_0 + m_1 = m.$$

Here, the inequality $|\mathcal{A}_1| \geq \omega_A$ holds, because the number of unknowns participating in any device is ω_A.
Case 2: If $m_1 \geq \omega_A$, from Claim 1 we can say,

$$|\mathcal{A}_0 \cup \mathcal{A}_1| \geq |\mathcal{A}_1| = k_A \geq m,$$

which concludes the proof of the lemma. ■

Example 2. Consider the same scenario in Example 1, where $k_A = 4$ and $s = 2$, therefore, $\mathcal{W}_0 = \{W_0, W_1, W_2, W_3\}$ and $\mathcal{W}_1 = \{W_4, W_5\}$. Now, choose $m = 3$ devices, W_0, W_1 and W_4. Thus, $m_0 = 2$ and $m_1 = 1$. Now, from the figure, we have $\mathcal{A}_0 = \{\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2\}$ and $\mathcal{A}_1 = \{\mathbf{A}_0, \mathbf{A}_1\}$. Hence, $|\mathcal{A}_0 \cup \mathcal{A}_1| = 3 \geq m$. Similar properties can be shown for any choice $m \leq k_A = 4$ different devices.

Now we state the following theorem which provides the guarantee of resilience to maximum number of stragglers for given storage constraints.

Theorem 1. Assume that a system has n edge devices each of which can store $1/k_A$ fraction of matrix $\mathbf{A}$ and the whole vector $\mathbf{x}$ for the distributed matrix-vector multiplication $\mathbf{A}^T\mathbf{x}$. If we assign the jobs according to Alg. 1, we achieve resilience to $s = n - k_A$ stragglers.

Proof. According to Alg. 1, first we partition matrix $\mathbf{A}$ into k_A disjoint block-columns. Thus, to recover the matrix-vector product, $\mathbf{A}^T\mathbf{x}$, we need to decode all k_A vector unknowns, $\mathbf{A}_0^T\mathbf{x}, \mathbf{A}_1^T\mathbf{x}, \mathbf{A}_2^T\mathbf{x}, \ldots, \mathbf{A}_{k_A-1}^T\mathbf{x}$. We denote the set of these k_A unknowns as $\mathcal{U}$. Now we choose an arbitrary set of k_A edge devices each of which corresponds to an equation in terms of ω_A of those k_A unknowns. Denoting the set of k_A equations as $\mathcal{V}$, we can say, $|\mathcal{U}| = |\mathcal{V}| = k_A$.

Now we consider a bipartite graph $\mathcal{G} = \mathcal{V} \cup \mathcal{U}$, where any vertex (equation) in $\mathcal{V}$ is connected to some vertices (unknowns) in $\mathcal{U}$ which participate in the corresponding equation. Thus, each vertex in $\mathcal{V}$ has a neighborhood of cardinality ω_A in $\mathcal{U}$.

Our goal is to show that there exists a perfect matching among the vertices of $\mathcal{V}$ and $\mathcal{U}$. To do so, we consider $\bar{\mathcal{V}} \subseteq \mathcal{V}$, where $|\bar{\mathcal{V}}| = m \leq k_A$. Now, we denote the neighbourhood of $\bar{\mathcal{V}}$ as $\mathcal{N}(\bar{\mathcal{V}}) \subseteq \mathcal{U}$. Thus, according to Lemma 1, for any $m \leq k_A$, we can say that $|\mathcal{N}(\bar{\mathcal{V}})| \geq m$. So, according to Hall's marriage theorem [55], we can say that there exists a perfect matching among the vertices of $\mathcal{V}$ and $\mathcal{U}$.

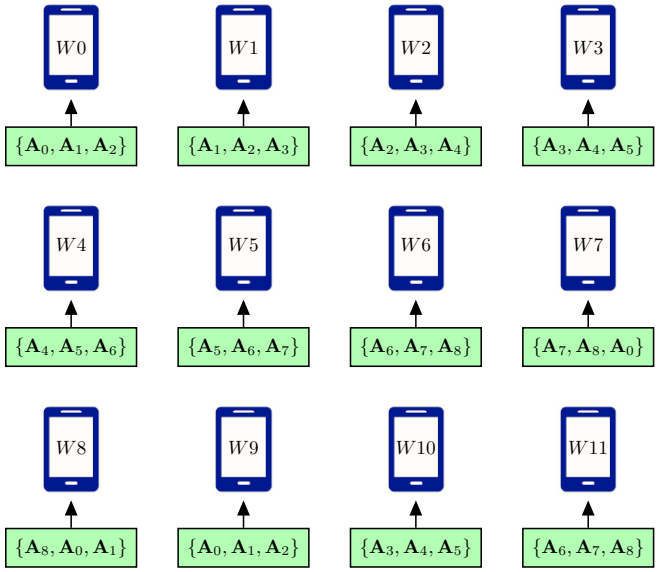

Fig. 2: Submatrix allocation for $n = 12$ workers and $s = 3$ stragglers, with $\gamma_A = \frac{1}{9}$ according to Alg. 1. Here, the weight of every submatrix is $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil = 3$. Any $\{\mathbf{A}_i, \mathbf{A}_j, \mathbf{A}_k\}$ indicates a random linear combination of the corresponding submatrices where the coefficients are chosen i.i.d. at random from a continuous distribution.

Next we consider the largest matching where the vertex $v_i \in \mathcal{V}$ is matched to the vertex $u_j \in \mathcal{U}$, which indicates that u_j participates in the equation corresponding to v_i. Now, considering k_A equations and k_A unknowns, we construct the $k_A \times k_A$ coding (or decoding) matrix $\mathbf{H}$ where row i corresponds to the equation associated to v_i where u_j participates. We replace row i of $\mathbf{H}$ by $\mathbf{e}_j$ where $\mathbf{e}_j$ is a unit row-vector of length k_A with the j-th entry being 1, and 0 otherwise. Thus we have a $k_A \times k_A$ matrix where each row has only one non-zero entry which is 1. In addition, since we have a perfect matching, $\mathbf{H}$ will have only one non-zero entry in every column. Thus, $\mathbf{H}$ is a permutation of the identity matrix, and therefore, $\mathbf{H}$ is full rank. Since the matrix is full rank for a choice of definite values, according to Schwartz-Zippel lemma [56], the matrix continues to be full rank for random choices of non-zero entries. Thus, the edge server can recover all k_A unknowns from any set of k_A edge devices. ■

Example 3. Consider a system with $n = 12$ devices each of which can store $1/9$-th fraction of matrix $\mathbf{A}$. We partition $\mathbf{A}$ as $\mathbf{A}_0, \mathbf{A}_1, \ldots, \mathbf{A}_8$. According to Alg. 1, we set the weight $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil = 3$, and assign random linear combinations of ω_A submatrices to each device as shown in Fig. 2. It can be verified that $\mathbf{A}^T\mathbf{x}$ can be recovered from *any* $\tau = k_A = 9$ devices, therefore, the scheme is resilient to *any* $s = 3$ stragglers.

Remark 1. Our proposed approach meets the lower bound on the weight for any $s \leq k_A$, as mentioned in Prop. 1. On the other hand, the approach in [31] always assigns a weight $\min(s + 1, k_A)$ which can be higher than ours when $s \leq k_A \leq s^2$ (e.g., Examples 1 and 3), and thus, may lead to reduction in worker computation speed compared to ours.

B. Extension to Heterogeneous System

In this section, we expand our approach described in Alg. 1 to accommodate a heterogeneous system comprising $\bar{n}$ edge devices, each with varying computational abilities and communication speeds. The assumption is that the storage capacities and processing speeds of the edge devices (i.e., the overall system architecture) are known before job assignment. Similar to the approach in [31], we assume that the system includes λ different types of devices, indexed from 0 to $\lambda-1$. For simplicity, we sort the devices in non-ascending order based on their types.

Let α represent the number of assigned columns and β the number of columns processed per unit time by the "weakest" type device [31]. In this setup, an edge device W_i of type j_i is allocated $c_{j_i}\alpha$ coded columns of the data matrix $\mathbf{A}$ and has a computation speed of $c_{j_i}\beta$, where $c_{j_i} \geq 1$ is an integer. A higher c_{j_i} indicates a "stronger" device W_i, capable of storing and processing data c_{j_i} times faster than the "weakest" type device.

Given that the devices are sorted in non-ascending order by type, we have $j_0 \geq j_1 \geq j_2 \geq \cdots \geq j_{\bar{n}-1} = 0$, which ensures $c_{j_0} \geq c_{j_1} \geq c_{j_2} \geq \cdots \geq c_{j_{\bar{n}-1}} = 1$. Note that when $\lambda = 1$ and all $c_{j_i} = 1$, the system reduces to the homogeneous case discussed in Section IV-A, where $0 \leq i \leq n-1$ and $j_i = 0$.

Consider the "weakest" type of edge device, which requires μ units of time to process α columns of $\mathbf{A}$. Consequently, any edge device W_i of type j_i can process $c_{j_i}\alpha$ columns within the same time frame, μ. From a computation and storage standpoint, this means that a node W_i (type j_i) can be viewed as the equivalent of $c_{j_i} \geq 1$ of the "weakest" type edge devices. Thus, the $\bar{n}$ devices in our heterogeneous system can be conceptualized as a homogeneous system composed of $n = \sum_{i=0}^{\bar{n}-1} c_{j_i}$ "weakest" type edge devices.

In other words, a device W_k in the heterogeneous system ($0 \leq k \leq \bar{n}-1$) can be represented as a combination of devices $\bar{W}_m, \bar{W}_{m+1}, \ldots, \bar{W}_{m+c_{kj}-1}$ in a homogeneous system, where $m = \sum_{i=0}^{k-1} c_{j_i}$, and W_k is of type j_i. To further illustrate this, for any worker node index $\bar{k}_A$ (where $0 \leq \bar{k}_A \leq \bar{n}-1$), we define $k_A = \sum_{i=0}^{\bar{k}A-1} c_{j_i}$ and $s = \sum_{i=\bar{k}_A}^{\bar{n}-1} c_{j_i}$, thus $n = \sum_{i=0}^{\bar{n}-1} c_{j_i} = k_A + s$. Thus, a heterogeneous system of $\bar{n}$ edge devices can be thought as a homogeneous system of $n = k_A + s$ nodes, for any $\bar{k}_A$ ($0 \leq \bar{k}_A \leq \bar{n}-1$).

Next, similar to [31], we state the following corollary (of Theorem 1) that demonstrates the straggler resilience property of our proposed approach in a heterogeneous setting. The corresponding proof can be obtained based on results in [31].

Corollary 2. Consider a heterogeneous system of $\bar{n}$ devices of different types and assume any $\bar{k}_A$ (where $0 \leq \bar{k}_A \leq \bar{n}-1$). Now, if the jobs are assigned to the modified homogeneous system of $n = k_A + s$ "weakest" type edge devices according to Alg. 1, the system will be resilient to s such nodes.

Example 4. Consider the example in Fig. 3, which involves $\bar{n} = 8$ edge devices. Suppose the computation capacities c_{j_i} are as follows: $c_{j_i} = 3$ when $i = 0$, $c_{j_i} = 2$ when $i = 1, 2$, and $c_{j_i} = 1$ for $3 \leq i \leq 7$. Therefore, the total computation capacity is $n = \sum_{i=0}^{\bar{n}-1} c_{j_i} = 12$. Now, $\bar{k}_A = 5$, leading to

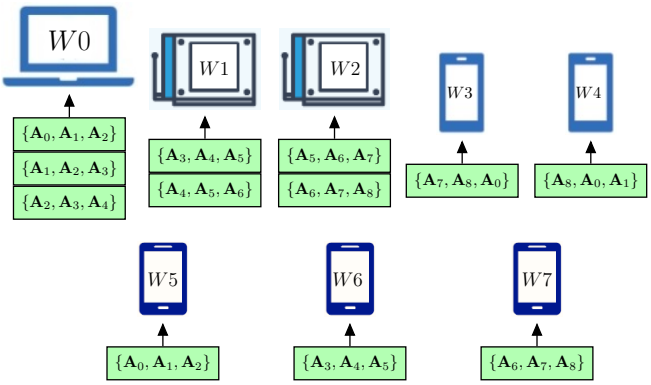

Fig. 3: A heterogeneous system where $\bar{n} = 8$ and $\bar{k}_A = 5$, and thus $n = 12$ and $k_A = 9$. First, W_0 is assigned thrice, and each of W_1 and W_2 is assigned twice the load of each of $W_3, W_4, \ldots, W_7$. This system is resilient to any $s = 3$ block-column processing, i.e., it is resilient to any three type 0 nodes (e.g., W_3 and W_6) or any one type 2 node (e.g., W_0).

$k_A = \sum_{i=0}^{\bar{k}A-1} c_{j_i} = 9$, and $s = \sum_{i=\bar{k}_A}^{\bar{n}-1} c_{j_i} = 3$. Hence, each "weakest" device is assigned $\frac{1}{9}$ of the total job. This scheme is resilient to the failure of any $s = 3$ block-column processing units, meaning it can tolerate the failure of any three type 0 devices or any one type 2 device.

If the "stronger" devices (those with $c_{j_i} > 1$) fail to complete all their tasks on time, our proposed system can still utilize their partial computations. For instance, consider a scenario where none of $W_3, W_4, \ldots, W_7$ is a straggler, but W_0, W_1 and W_2 are slower than their expected speed. If W_0 completes two out of its three assigned tasks, and each of W_1 and W_2 completes one out of their two assigned tasks, we can still recover the final result. This is because we only need to process $k_A = 9$ block-columns across all nodes. Thus, our approach can leverage partial stragglers (W_0, W_1, and W_2 in this example).

Remark 2. While we extend our method to the heterogeneous case in a similar manner as [31], our scheme involves a smaller encoding weight (ω) compared to [31], which enhances the overall speed of the job for a sparse input matrix $\mathbf{A}$.

C. Computational Complexity for a edge device

In this work, we assume that the "input" matrix, $\mathbf{A} \in \mathbb{R}^{t \times r}$, is sparse, i.e., most of the entries of $\mathbf{A}$ are zero. Let us assume that the probability for any entry of $\mathbf{A}$ to be non-zero is μ, where $\mu > 0$ is very small. According to Alg. 1, we combine ω_A submatrices (of size $t \times r/k_A$) to obtain the coded submatrices and assign them to the edge devices. Hence, the probability for any entry of any coded submatrix to be non-zero is $1-(1-\mu)^{\omega_A}$ which can be approximated by $\omega_A\mu$. Thus, in our approach, the per edge device computational complexity is $\mathcal{O}\left(\omega_A\mu \times \frac{rt}{k_A}\right)$ where $\omega_A = \left\lceil \frac{k_A(s+1)}{k_A+s} \right\rceil$.

On the other hand, the dense coded approaches [25], [32], [33] combine k_A submatrices for encoding, hence, their per edge device computational complexity is $\mathcal{O}\left(k_A\mu \times \frac{rt}{k_A}\right) = \mathcal{O}\left(\mu \times rt\right)$ which is $\frac{k_A}{\omega_A} \approx \frac{s+k_A}{s+1}$ times higher than that of ours. Moreover, the recent sparse matrix computations approach in [31] combines $s+1$ submatrices for encoding (when

$s < k_A$). Thus, its corresponding computational complexity is $\mathcal{O}\left((s+1)\mu \times \frac{rt}{k_A}\right)$; approximately $(1+s/k)$ times higher than that of ours. We clarify this with the following example.

Example 5. Consider the same setting in Example 3 where $n = 12$, $k_A = 9$ and $s = 3$. In this scenario, the recent work [31] assigns random linear combinations of $\min(s+1, k_A) = 4$ submatrices to each device. Thus, our proposed approach enjoys a 25% decrease in computational complexity, which could significantly enhance the overall computational speed.

D. Numerical Stability and Coefficient Determination Time

In this section, we discuss the numerical stability of our proposed matrix-vector multiplication scheme. The condition number is widely regarded as a significant measure of numerical stability for such a system [23], [32], [33]. In the context of a system consisting of n edge devices with s stragglers, the worst-case condition number (κ_{worst}) is defined as the highest condition number among the decoding matrices when considering all possible choices of s stragglers. In methods involving random coding like ours, the idea is to generate random coefficients multiple (e.g., 20) times and selecting the set of coefficients that results in the lowest κ_{worst}.

In our proposed method, we partition matrix $\mathbf{A}$ into k_A disjoint block-columns, which underscores the necessity to recover k_A vector unknowns. Consequently, in each attempt, we must determine the condition numbers of $\binom{n}{k_A}$ decoding matrices, each of size $k_A \times k_A$. This whole process has a total complexity of $\mathcal{O}\left(\binom{n}{k_A} k_A^3\right)$. On the other hand, the recent sparse matrix computation techniques, such as sparsely coded straggler (SCS) optimal scheme discussed in [36] or the class-based scheme discussed in [29] partition matrix $\mathbf{A}$ into $\Delta_A = \text{LCM}(n, k_A)$ block-columns. Thus, in each attempt, they need to ascertain the condition numbers of $\binom{n}{k_A}$ matrices, each of which has a size $\Delta_A \times \Delta_A$, resulting in a total complexity of $\mathcal{O}\left(\binom{n}{k_A} \Delta_A^3\right)$. Since Δ_A can be considerably larger than k_A, those methods involve significantly more complexity compared to our proposed scheme.

Remark 3. We can also utilize our proposed approach in a D2D-enabled Federated Learning scenario as discussed in [10]. The assumption is that the edge devices may be responsible for generating local data and also for performing some computational tasks. The devices transmit their local submatrices to some other trusted devices, and build resilience against the computationally slower devices. Since our proposed approach involves a smaller weight, any device would require transmitting to less number of other devices compared to the approach in [10], therefore, the overall process would be faster.

V. Proposed Matrix-matrix Multiplication Approach

In this section, we discuss our proposed distributed matrix-matrix multiplication approach. As we mentioned above, we consider a system of n edge devices, each of which can store $1/k_A$ and $1/k_B$ fraction of matrices $\mathbf{A}$ and $\mathbf{B}$, respectively, and our aim is to build resilience to any $s = n - k$ stragglers,

Algorithm 2: Proposed scheme for distributed matrix-matrix multiplication

Input : Matrices $\mathbf{A}$ and $\mathbf{B}$, n-number of edge devices, storage fraction $\gamma_A = \frac{1}{k_A}$ and $\gamma_A = \frac{1}{k_B}$ (w.l.o.g., $k_A \le k_B$), and set $k = k_A k_B$; number of stragglers, $s \le k$.

Partition matrices $\mathbf{A}$ and $\mathbf{B}$ into k_A and k_B block-columns, respectively;

Create a $n \times k_A$ random matrix $\mathbf{R}_A$ with entries $r^A_{i,j}$, $0 \le i \le n-1$ and $0 \le j \le k_A - 1$;

Create a $n \times k_B$ random matrix $\mathbf{R}_B$ with entries $r^B_{i,j}$, $0 \le i \le n-1$ and $0 \le j \le k_B - 1$;

Set weights $\omega_A | k_A$ and $\omega_B | k_B$ (where $\omega_A \le \omega_B$) so that $\omega_A \omega_B \ge \left\lceil \frac{k_A k_B (s+1)}{k_A k_B + s} \right\rceil$ with $1 < \omega_A < k_A$;

for $i \leftarrow 0$ **to** $n-1$ **do**

  if $i < k_A k_B$ **then**

    Define $T = \{i, i+1, \ldots, i+\omega_A - 1\}$ (entries reduced modulo k_A);

    Define $S = \{j, j+1, \ldots, j+\omega_B - 1\}$ (entries reduced modulo k_B) where $j = \lfloor i/k_A \rfloor$;

  else

    $T = \{\ell\omega_A, \ell\omega_A + 1, \ldots, (\ell+1)\omega_A - 1\}$ (reduced modulo k_A) when $\ell = \text{mod}(i, k_A)$;

    $S = \{m\omega_B, m\omega_B + 1, \ldots, (m+1)\omega_B - 1\}$ (reduced modulo k_B) where $m = \left\lfloor \frac{i\,\omega_A}{k_A} \right\rfloor$;

  end

  Create a random linear combination of $\mathbf{A}_q$'s where $q \in T$, thus $\tilde{\mathbf{A}}_i = \sum_{q \in T} r^A_{i,q} \mathbf{A}_q$;

  Create a random linear combination of $\mathbf{B}_q$'s where $q \in S$, thus $\tilde{\mathbf{B}}_i = \sum_{q \in S} r^B_{i,q} \mathbf{B}_q$;

  The edge server assigns encoded submatrices $\tilde{\mathbf{A}}_i$ and $\tilde{\mathbf{B}}_i$ to edge device W_i;

  Edge device W_i computes $\tilde{\mathbf{A}}_i^T \tilde{\mathbf{B}}_i$;

end

Output : The edge server recovers $\mathbf{A}^T\mathbf{B}$ from the fastest $k_A k_B$ edge devices.

where $k = k_A k_B$. Similar to the matrix-vector case, we assume that $s \le k$, i.e., not more than half of the devices will not be stragglers. The overall procedure is given in Alg. 2.

Here we provide an overview of our proposed method. First we partition matrices $\mathbf{A}$ and $\mathbf{B}$ into k_A and k_B disjoint block columns, respectively, as $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_{k_A-1}$ and $\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_{k_B-1}$, respectively. Then we consider the lower bound stated in Prop. 1, and find $\hat{\omega} = \lceil \frac{(n-s)(s+1)}{n} \rceil$. Next, we find an $\omega \ge \hat{\omega}$ such that $\omega = \omega_A \omega_B$ where $1 < \omega_A < k_A$ and $1 < \omega_B < k_B$, $\omega_A | k_A$ and $\omega_B | k_B$.

Now, within the edge devices in $\mathcal{W}_0 = \{W_0, W_1, \ldots, W_{k-1}\}$, we assign a random linear combination of $\mathbf{A}_i, \mathbf{A}_{i+1}, \ldots, \mathbf{A}_{i+\omega_A-1}$ (indices of $\mathbf{A}$ are reduced modulo k_A) to edge device W_i, $0 \le i \le k-1$. In other words, the corresponding submatrices of $\mathbf{A}$ are shifted in a cyclic manner over the edge devices in $\mathcal{W}_0$. Next we set $j = \lfloor i/k_A \rfloor$, and assign $\mathbf{B}_j, \mathbf{B}_{j+1}, \ldots, \mathbf{B}_{j+\omega_B-1}$ (indices of $\mathbf{B}$ are reduced modulo k_B) to edge device W_i, where $0 \le i \le k-1$.

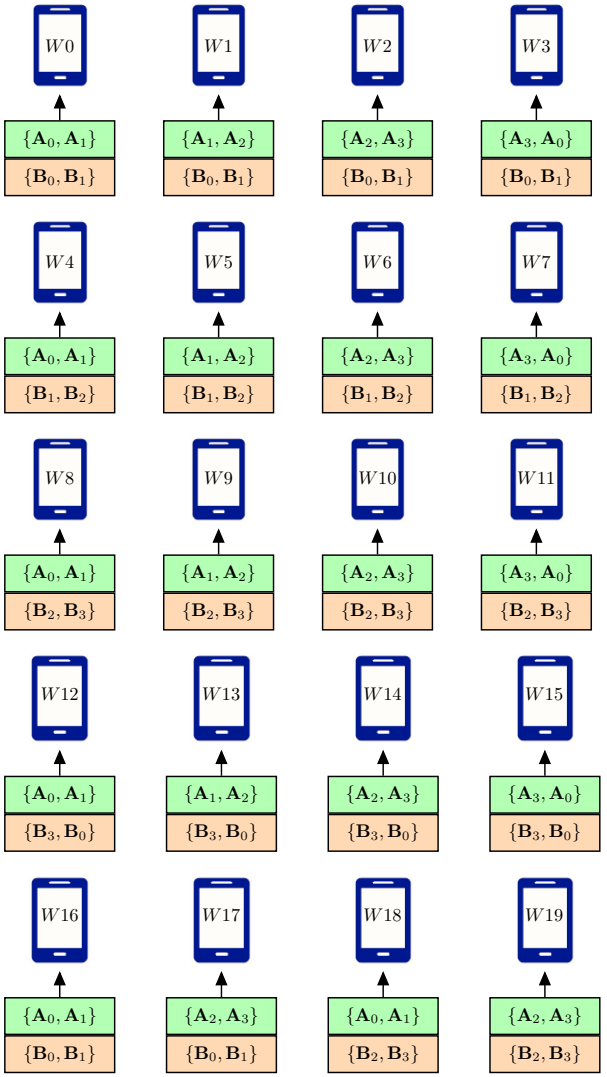

Fig. 4: Submatrix allocation according to Alg. 2 when $n = 20$ with $\gamma_A = \gamma_B = \frac{1}{4}$; thus resilient to $s = n - \frac{1}{\gamma_A \gamma_B} = 4$ straggler devices. The weights of the submatrices are $\omega_A = \omega_B = 2$. Any assignment $\{\mathbf{A}_i, \mathbf{A}_j\}$ or $\{\mathbf{B}_i, \mathbf{B}_j\}$ indicates a random linear combination of the corresponding submatrices where the coefficients are chosen i.i.d. at random from a continuous distribution.

After that, within the edge devices in $\mathcal{W}_1 = \{W_k, W_{k+1}, \ldots, W_{n-1}\}$, we assign a random linear combination of $\mathbf{A}_{\ell\omega_A}, \mathbf{A}_{\ell\omega_A+1}, \ldots, \mathbf{A}_{(\ell+1)\omega_A-1}$ (indices of $\mathbf{A}$ are reduced modulo k_A) to edge device W_i, where $\ell = \text{mod}(i, k_A)$ and $k \leq i \leq n-1$. Next we set $m = \lfloor \frac{i\omega_A}{k_A} \rfloor$, and assign $\mathbf{B}_{m\omega_B}, \mathbf{B}_{m\omega_B+1}, \ldots, \mathbf{B}_{(m+1)\omega_B-1}$ (indices of $\mathbf{B}$ are reduced modulo k_B) to edge device W_i, where $k \leq i \leq n-1$. After assigning the jobs, all the edge devices will start computing their respective submatrix-products, and once the fastest $\tau = k_A k_B$ devices finish and return their results, the edge server recovers all the unknowns in the form of $\mathbf{A}_i^T \mathbf{B}_j$, where $0 \leq i \leq k_A - 1$ and $0 \leq j \leq k_B - 1$.

Example 6. Consider the example shown in Fig. 4 where $n = 20, \gamma_A = \gamma_B = 1/4$. So, we partition $\mathbf{A}$ and $\mathbf{B}$ into $k_A = k_B = 4$ block-columns, respectively. In each of the devices, according to Alg. 2, we assign one coded submatrix from $\mathbf{A}$ and one from $\mathbf{B}$ which are linear combinations of $\omega_A = \omega_B = 2$ uncoded submatrices with coefficients chosen i.i.d. at random from a continuous distribution. It can be verified that this scheme is resilient to $s = n - k_A k_B = 4$ stragglers.

1) Structure of the Job Assignment: To describe the structure of the proposed scheme, first we partition the edge devices in $\mathcal{W}_0 = \{W_0, W_1, W_2, \ldots, W_{k-1}\}$ into k_A disjoint classes, denoted by $\mathcal{M}_i$'s, where any $\mathcal{M}_i$ consists of all the edge devices W_j's if $j \equiv i (\text{mod } k_A)$. In other words, $\mathcal{M}_i = \{W_i, W_{k_A+i}, W_{2k_A+i}, \ldots\}$, for $i = 0, 1, 2, \ldots, k_A - 1$. Hence, $|\mathcal{M}_i| = k_B$, for any i.

Moreover, according to our proposed scheme, the participating submatrices of $\mathbf{A}$ are the same over all the edge devices in any $\mathcal{M}_i$. For instance, in Fig. 4, we have $\mathcal{M}_0 = \{W_0, W_4, W_8, W_{12}\}$, and random linear combinations of $\mathbf{A}_0$ and $\mathbf{A}_1$ are assigned to all the corresponding edge devices. At this point, we define a set, $\mathcal{D}_i^A = \{\mathbf{A}_i, \mathbf{A}_{i+1}, \ldots, \mathbf{A}_{i+\omega_A-1}\}$, which consists of the participating submatrices of $\mathbf{A}$ corresponding to edge device set $\mathcal{M}_i$, where the indices are reduced modulo k_A. Now we state the following claim (with the proof in [31]) which gives a lower bound for the cardinality of the union of any arbitrary number of $\mathcal{D}_i^A$'s.

Claim 2. Consider any q sets $\mathcal{D}_i^A$'s, $q \leq k_A - \omega_A + 1$, denoted w.l.o.g., $\bar{\mathcal{D}}_j^A$, $0 \leq j \leq q-1$ arbitrarily. Then $\left| \bigcup_{j=0}^{q-1} \bar{\mathcal{D}}_j^A \right| \geq \omega_A + q - 1$.

Now, consider any particular $\mathcal{M}_i$. According to Alg. 2, the participating submatrices of $\mathbf{B}$ are shifted in a cyclic fashion over the edge devices of any $\mathcal{M}_i$. For instance, in Example 6, the participating submatrices, $\mathbf{B}_0, \mathbf{B}_1, \mathbf{B}_2$ and $\mathbf{B}_3$, are shifted in a cyclic fashion within the edge devices of $\mathcal{M}_0$, i.e., W_0, W_4, W_8 and W_{12}. Next, in the following claim, we find the minimum number of participating unknowns (in the form of $\mathbf{A}_u^T \mathbf{B}_v$) within any δ edge devices from any $\mathcal{M}_q$.

Claim 3. Consider $\mathcal{M}_q$, $0 \leq q \leq k_A - 1$. Denote the minimum of total number of participating unknowns (in the form of $\mathbf{A}_i^T \mathbf{B}_j$) within any δ edge devices from $\mathcal{M}_q$ by ρ. Then

$$\rho = \omega_A \times \min\left(\omega_B + \delta - 1, k_B\right) .$$

Proof. The participating uncoded submatrices of $\mathbf{B}$ are shifted in a cyclic fashion within the edge devices of $\mathcal{M}_q$. Thus according to the proof of cyclic scheme in Appendix C in [36], the minimum number of constituent submatrices of $\mathbf{B}$ within any δ edge devices of $\mathcal{M}_q$ is $\min\left(\omega_B + \delta - 1, k_B\right)$ if $\delta \geq 1$. Now the coded submatrices of $\mathbf{B}$ are multiplied by linear combinations of the same ω_A submatrices, thus the minimum of total number of participating unknowns (in the form of $\mathbf{A}_u^T \mathbf{B}_v$) is $\rho = \omega_A \times \min\left(\omega_B + \delta - 1, k_B\right)$. ■

2) Rearrangement of $\mathcal{M}_i$'s: Before stating the necessary theorem, we discuss a pre-processing step that rearranges the $\mathcal{M}_i$'s. Choose any arbitrary m_0 edge devices ($m_0 \leq k_A k_B$), and assume that δ_i devices have been chosen from $\mathcal{M}_i$, for $0 \leq i \leq k_A - 1$, so that $\sum_{i=0}^{k_A-1} \delta_i = m_0$. Now, we rearrange the δ_i's in a non-increasing sequence so that $\tilde{\delta}_0 \geq \tilde{\delta}_1 \geq \tilde{\delta}_2 \geq \cdots \geq \tilde{\delta}_{k_A-1}$, and rename the corresponding $\mathcal{M}_i$'s as $\tilde{\mathcal{M}}_i$'s so that $\tilde{\delta}_i$ devices have been chosen from $\tilde{\mathcal{M}}_i$.

Now, we denote ρ_0 as the minimum of total number of participating unknowns (in the form of $\mathbf{A}_i^T\mathbf{B}_j$) within the $\tilde{\delta}_0$ edge devices of $\tilde{\mathcal{M}}_0$. Thus, according to Claim 3,

$$\rho_0 = \omega_A \times \min(\omega_B + \tilde{\delta}_0 - 1, k_B) \ . \tag{4}$$

After that, we move to $\tilde{\mathcal{M}}_1, \tilde{\mathcal{M}}_2, \ldots, \tilde{\mathcal{M}}_{k_A-\omega_A}$, sequentially, to find the number of additional participating unknowns within the corresponding $\tilde{\delta}_i$ edge devices of $\tilde{\mathcal{M}}_i$, where $1 \leq i \leq k_A-1$. We denote ρ_i as the minimum number of such additional participating unknowns in $\tilde{\mathcal{M}}_i$.

Here, according to Claim 2, $\left|\bigcup_{j=0}^{0} \bar{\mathcal{D}}_j^A\right| \geq \omega_A$ and $\left|\bigcup_{j=0}^{1} \bar{\mathcal{D}}_j^A\right| \geq \omega_A + 1$. Thus, there will be at least one additional participating submatrix of $\mathbf{A}$ in $\tilde{\mathcal{M}}_0 \cup \tilde{\mathcal{M}}_1$ in comparison to $\tilde{\mathcal{M}}_0$, and the property will continue to hold until we consider the set $\tilde{\mathcal{M}}_0 \cup \tilde{\mathcal{M}}_1 \cup \cdots \cup \tilde{\mathcal{M}}_0 \cup \tilde{\mathcal{M}}_{k_A-\omega_A}$. Now, since the submatrices of $\mathbf{B}$ (which will be multiplied by the additional submatrix of $\mathbf{A}$) are shifted in a cyclic fashion within any $\tilde{\mathcal{M}}_i$,

$$\rho_i = \min\left(\omega_B + \tilde{\delta}_i - 1, k_B\right) \ , \tag{5}$$

for $1 \leq i \leq k_A - \omega_A$. Note that, ρ_i has a trivial lower bound, *zero*, when $k_A - \omega_A + 1 \leq i \leq k_A - 1$.

Now we state the following lemma that provides a lower bound on the minimum number of participating unknowns (in the form of $\mathbf{A}_u^T\mathbf{B}_v$) in the equations from any arbitrary $m_0 \leq k_A k_B$ devices. Note that, we assume $k_B \geq k_A$ without loss of generality; if $k_A > k_B$, we can compute $\mathbf{A}^T\mathbf{B}$ as $\left(\mathbf{B}^T\mathbf{A}\right)^T$ without any additional computational cost.

Lemma 2. For any arbitrary $k_A \geq 3$ and $k_B \geq 3$ (where $k_A \leq k_B$ without loss of generality), if we assign the jobs to $n = k + s$ edge devices (where $s \leq k = k_A k_B$) according to Alg. 2, then the minimum of total number of participating unknowns within any m edge devices ($m \leq k$) will be lower bounded by m.

Proof. We prove the lemma in a similar manner as we have proved Lemma 1. First, we partition all n edge devices into *two* sets where $\mathcal{W}_0$ includes the first k devices and $\mathcal{W}_1$, includes the next s edge devices, i.e., we have $\mathcal{W}_0 = \{W_0, W_1, W_2, \ldots, W_{k-1}\}$ and $\mathcal{W}_1 = \{W_k, W_{k+1}, \ldots, W_{n-1}\}$, where $k = k_A k_B$. Thus, we have $|\mathcal{W}_0| = k$ and $|\mathcal{W}_1| = s \leq k$. Now, we choose any $m \leq k$ edge devices, where we choose m_0 devices from $\mathcal{W}_0$ and m_1 devices from $\mathcal{W}_1$, so that $m = m_0 + m_1$. We denote set of the participating unknowns within those devices as $\mathcal{X}_0$ and $\mathcal{X}_1$, respectively. Hence, to prove the lemma, we need to show $|\mathcal{X}_0 \cup \mathcal{X}_1| \geq m$, for any $m \leq k$.

Now we state the following claims to find the number of participating unknowns in $\mathcal{W}_0$ and $\mathcal{W}_1$, with the proofs in App. A and App. B, respectively..

Claim 4. Choose any $m_0 \leq k$ devices from $\mathcal{W}_0$ such that $m_0 > 0$. The number of participating submatrices within these devices is lower bounded by $\min(m_0 + \omega - 1, k)$.

Claim 5. Choose any $m_1 \geq \omega$ devices from $\mathcal{W}_1$. The number of participating submatrices within these devices, $|\mathcal{X}_1| = k$.

Case 1: $m_1 \leq \omega - 1$. In this case, from Claim 4, we have

$$\begin{aligned} |\mathcal{X}_0 \cup \mathcal{X}_1| \geq |\mathcal{X}_0| &= \min(m_0 + \omega - 1, k) \ \ (\text{when } m_0 > 0) \\ &\geq \min(m_0 + m_1, k) \geq m, \end{aligned}$$

since $m = m_0 + m_1 \leq k$. We take account the remaining scenario when $m_0 = 0$. In that case,

$$|\mathcal{X}_0 \cup \mathcal{X}_1| \geq |\mathcal{X}_1| \geq \omega \geq m_0 + m_1 = m.$$

Here, the inequality $|\mathcal{X}_1| \geq \omega = \omega_A\omega_B$ holds, because the number of unknowns participating in any device is $\omega_A\omega_B$.

Case 2: $m_1 \geq \omega$. In this case, from Claim 5 we can say,

$$|\mathcal{X}_0 \cup \mathcal{X}_1| \geq |\mathcal{X}_1| = k \geq m,$$

which concludes the proof of the lemma. ■

While Alg. 2 and Lemma 2 have been developed and proved for scenarios where $\omega_A | k_A$ and $\omega_B | k_B$, our approach is applicable for other values also. The job assignment in the edge devices can still be the same as mentioned in Alg. 2.

Example 7. Consider a distributed system when $k > s^2$ as mentioned in Corollary 1(i). In this scenario, $\hat{\omega} = s + 1$. However, $|\mathcal{W}_1| = s$, thus we have $m_1 \leq \omega - 1$. It indicates that only Case 1 is enough to conclude the proof of Lemma 2 in this setting. Hence, we do not require the constraints $\omega_A | k_A$ and $\omega_B | k_B$ here, which were required only to prove Claim 5 (and hence, the corresponding Case 2).

Now we state the following theorem which provides the guarantee of resilience to maximum number of stragglers for given storage constraints.

Theorem 2. Assume that a system has n edge devices each of which can store $1/k_A$ and $1/k_B$ fractions of matrices $\mathbf{A}$ and $\mathbf{B}$ respectively, for distributed matrix-matrix multiplication $\mathbf{A}^T\mathbf{B}$; $k_A \leq k_B$. If we assign the jobs according to Alg. 2, we achieve resilience to $s = n - k_A k_B$ stragglers, where $s \leq k_A k_B$.

Proof. According to Alg. 2, first we partition matrix $\mathbf{A}$ and $\mathbf{B}$ into k_A and k_B disjoint block-columns, respectively. Thus, to recover the matrix-matrix product, $\mathbf{A}^T\mathbf{B}$, we need to decode all $k = k_A k_B$ matrix unknowns, in the form of $\mathbf{A}_i^T\mathbf{B}_j$, where $0 \leq i \leq k_A - 1$ and $0 \leq j \leq k_B - 1$. We denote the set of these k unknowns as $\mathcal{U}$. Now we choose an arbitrary set of $k_A k_B$ edge devices each of which corresponds to an equation in terms of $\omega = \omega_A\omega_B$ of those k unknowns. Denoting the set of k equations as $\mathcal{V}$, we can say, $|\mathcal{U}| = |\mathcal{V}| = k$.

Now, similar to the proof of Theorem 1, we consider a bipartite graph $\mathcal{G} = \mathcal{V} \cup \mathcal{U}$, where any vertex (equation) in $\mathcal{V}$ is connected to some vertices (unknowns) in $\mathcal{U}$ which participate in the corresponding equation. Thus, each vertex in $\mathcal{V}$ has a neighborhood of cardinality ω in $\mathcal{U}$. Our goal is to show that there exists a perfect matching among the vertices of $\mathcal{V}$ and $\mathcal{U}$. Now, according to Lemma 2, for any $m \leq k$, we can say that $|\mathcal{N}(\bar{\mathcal{V}})| \geq m$. Thus, similar to to the proof of Theorem 1, according to Hall's marriage theorem [55] and Schwartz-Zippel lemma [56], we can prove that the edge server can recover all $k = k_A k_B$ unknowns from any set of k edge devices. ■

Remark 4. While Theorem 2 has been developed considering a homogeneous setting of edge devices, it can be also extended

to a heterogeneous setting [31], similar to the matrix-vector case discussed in Sec. IV-B.

A. Computational Complexity for a Worker Node

In this work, we assume that $\mathbf{A} \in \mathbb{R}^{t\times r}$ and $\mathbf{B} \in \mathbb{R}^{t\times w}$ are sparse, only a small fraction (η) of the entries are non-zero. Therefore, in our case, the computational complexity for any edge device is $\mathcal{O}\left(\omega_A\eta \times \omega_B\eta \times t \times \frac{rw}{k_Ak_B}\right) = \mathcal{O}\left(\omega_A\omega_B\eta^2 \times \frac{rwt}{k_Ak_B}\right)$. On the other hand, any dense coded approach [25], [32] assigns linear combinations of k_A and k_B submatrices, hence, the corresponding computational complexity is approximately $\mathcal{O}\left(k_A\eta \times k_B\eta \times t \times \frac{rw}{k_Ak_B}\right) = \mathcal{O}\left(\eta^2 \times rwt\right)$; $\frac{k_Ak_B}{\omega_A\omega_B}$ times larger than ours, as $\omega_A < k_A, \omega_B < k_B$.

Example 8. Consider the example in Fig. 4, where $k_A = k_B = 4$ and $s = 4$. The approach in [31] requires $\omega \geq 5$; hence sets $\omega_A = 3$ and $\omega_B = 2$. On the other hand, in our approach, we require $\omega = \left\lceil \frac{16\times5}{20} \right\rceil = 4$, hence, we set $\omega_A = \omega_B = 2$. This indicates a 33% reduction of computational complexity in our case compared to the method in [31].

B. Numerical Stability

Similar to the discussion in the matrix-vector case in Sec. IV-D, we can develop a numerically stable scheme using Alg. 2. Note that in the matrix-matrix case, we need to find two "good" sets of random coefficients, one for the encoding of $\mathbf{A}$, and the other for the encoding of $\mathbf{B}$. We empirically demonstrate the numerical stability of our approach in Sec. VI.

VI. Numerical Experiments

In this section, we conduct simulations on distributed matrix-computations, which can be incorporated within the framework of numerous data processing tasks. For example, high-dimensional linear transformations are vital for dimensionality reduction techniques such as principal component analysis (PCA) [57] and linear discriminant analysis (LDA) [58]. Furthermore, they are fundamental to training deep neural networks and employing them for classification. For example, every layer of a fully-connected deep neural network necessitates matrix-matrix multiplications during both forward and backward propagation [4].

For our numerical experiments, we compare the performance of our algorithm against various alternative techniques [25], [29], [31]–[33], [36]. It is important to note the existence of several other methodologies tailored specifically for sparse matrix computations. Notably, the approach outlined in [34] lacks resilience to the maximum number of stragglers given certain storage constraints. Additionally, the strategy proposed in [35], which partitions edge devices into untrusted and partly trusted clusters, diverges from our foundational assumptions. The approach described in [59], which delegates certain tasks to the edge server to mitigate the probability of rank-deficiency during decoding, also does not align with our model assumptions. Therefore, these methodologies are excluded from consideration in our numerical experimentation section.

We explore a distributed system that consists of $n = 42$ edge devices with $s = 6$ stragglers. We assume that each device can store $\gamma_A = \gamma_B = \frac{1}{6}$ fraction of matrices $\mathbf{A}$ and $\mathbf{B}$. We consider sparse input matrices $\mathbf{A}$ of size $20,000 \times 15000$ and $\mathbf{B}$ of size $20,000 \times 12000$. We assume three different cases where the sparsity of $\mathbf{A}$ and $\mathbf{B}$ are 95%, 98% and 99%, respectively, which indicate that randomly chosen 95%, 98% and 99% entries of the matrices are zero. It is worth noting that there exist numerous practical instances where data matrices demonstrate such (or, even more) levels of sparsity (refer to [49] for specific examples). We carry out the experiments on an AWS (Amazon Web Services) cluster, utilizing a `c5.18xlarge` machine as the server and `t2.small` machines as the edge devices.

Worker computation time: Table II presents a comparison among different methods based on the computation time required by edge devices to complete their respective tasks. In this scenario, where $k_A = k_B = 6$, the approaches described in [25], [32], [33] allocate two linear combinations of $k_A = k_B = 6$ submatrices of $\mathbf{A}$ and $\mathbf{B}$, respectively, to each of the edge devices. Consequently, the original sparsity of matrix $\mathbf{A}$ (and $\mathbf{B}$) is lost within the encoded submatrices. As a result, the edge devices experience a significantly increased processing time for their tasks compared to our proposed approach or the methods outlined in [29], [31], [36], which are specifically designed for sparse matrices and involve smaller weights.

To discuss the effectiveness of our approach in more details, we compare the weight of the coding of our approach against the approach in [31]. In this scenario, when $n = 42$ and $s = 6$, our approach sets the weight $\left\lceil \frac{(n-s)(s+1)}{n} \right\rceil = \left\lceil \frac{36\times7}{42} \right\rceil = 6$; thus we set $\omega_A = 2$ and $\omega_B = 3$. On the other hand, the approach in [10] uses a weight greater than or equal to $s+1 = 7$, thus needs to set $\omega_A = 4$ and $\omega_B = 2$. Hence, our approach involves around 25% less computational complexity per edge device, which is supported by the results in Table II.

Communication delay: In Table II, we also illustrate the delay incurred during the transmission of encoded submatrices from the server to the edge devices. The methods outlined in [25], [32], and [33] utilize dense linear combinations of submatrices, causing a notable rise in non-zero entries within these encoded submatrices. Consequently, transmitting this increased number of non-zero entries results in significant communication delays. In contrast, our proposed approach addresses this issue by employing encoded submatrices formed through linear combinations of a limited number of uncoded submatrices, thus markedly reducing communication delays.

In addition, here we compare our proposed approach against the method in [31] in terms of the approximate number of non-zero entries that need to be sent from the edge server to any edge device. We consider the case when the matrices $\mathbf{A}$ and $\mathbf{B}$ have approximately 99% entries to be zero, hence, only 1% entries are non-zero. Now, in this scenario ($n = 42, k_A = k_B = 6$ and $s = 6$), the corresponding number of non-zero entries for our proposed approach is approximately $\frac{20k\times15k}{k_A} \times 0.01 \times \omega_A + \frac{20k\times12k}{k_B} \times 0.01 \times \omega_B = 2.2 \times 10^6$, since we set $\omega_A = 2$ and $\omega_B = 3$. On the other hand, the number of the non-zero entries to be sent to each edge device in the approach [31] is approximately $\frac{20k\times15k}{k_A} \times 0.01 \times \omega_A + \frac{20k\times12k}{k_B} \times 0.01 \times \omega_B = 2.8 \times 10^6$, since in that case $\omega_A = 4$ and $\omega_B = 2$. Thus our proposed approach requires the server

TABLE II: Comparison of worker computation time and communication delay for matrix-matrix multiplication for $n = 42, \gamma_A = \gamma_B = \frac{1}{6}$ when randomly chosen 95%, 98% and 99% entries of matrices $\mathbf{A}$ and $\mathbf{B}$ are zero.

Methods	Worker Comp. Time (in s)			Communication Delay (in s)		
	$\mu = 99\%$	$\mu = 98\%$	$\mu = 95\%$	$\mu = 99\%$	$\mu = 98\%$	$\mu = 95\%$
Poly. Code [25]	1.65	5.19	8.95	0.78	1.42	2.44
Ortho Poly Code [32]	1.60	5.16	9.03	0.80	1.48	2.36
RKRP Code [33]	1.61	5.11	8.96	0.82	1.44	2.39
SCS Optimal Scheme [36]	1.09	2.12	5.23	0.39	0.58	0.93
Class-based Scheme [29]	0.74	1.21	4.12	0.30	0.47	0.77
Cyclic Code [31]	0.76	1.24	4.19	0.32	0.51	0.83
Proposed Scheme	**0.61**	**1.04**	**3.11**	**0.25**	**0.39**	**0.65**

TABLE III: Comparison among different approaches in terms of worst case condition number (κ_{worst}) and the corresponding required time for 10 trials to find a good set of random coefficients

Methods	κ_{worst} for $n = 42$, $s = 6$	Req. time for 10 trials
Poly. Code [25]	3.67×10^{23}	0
Ortho-Poly [32]	8.80×10^{10}	0
RKRP Code [33]	4.84×10^{8}	84 min
SCS Opt. Sch. [36]	7.88×10^{9}	16 hr
Class based [29]	5.81×10^{8}	15 hr
Cyclic Code [31]	2.13×10^{9}	85 min
Proposed Scheme	$\mathbf{7.95 \times 10^{8}}$	**86** min

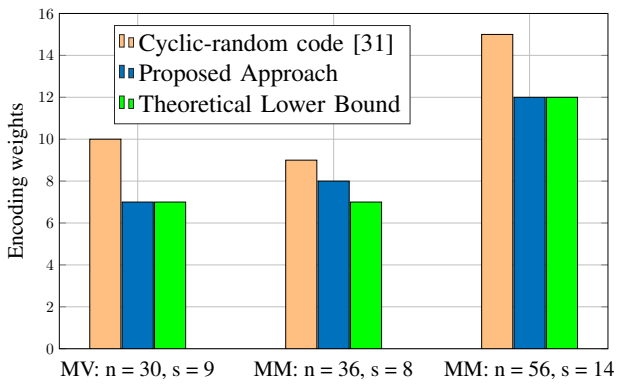

Fig. 5: Comparison of encoding weights in matrix-vector (MV) and matrix-matrix (MM) multiplication between the method in [31], our proposed approach, and the theoretical lower bound for different choices of n and s.

to transmit approximately 22% less non-zero entries than the method in [31]; Table II confirms this gain of our approach.

Numerical stability: Next, we assess the numerical stability of distributed systems using different coded matrix computation techniques. We examine the condition numbers of the decoding matrices for various combinations of $n = 42$ workers and $s = 6$ stragglers. By comparing the worst-case condition number (κ_{worst}) across different methods, we present the κ_{worst} values in Table III. The polynomial coding method [25] encounters issues with ill-conditioned Vandermonde matrices, leading to pronounced numerical instability, as indicated by its notably elevated κ_{worst} value. In contrast, our proposed technique, characterized by numerical robustness, yields a smaller κ_{worst} compared to the method described in [32] where the condition numbers grow exponentially in terms of $s = n-k$. Although the approach outlined in [33] achieves a slightly reduced κ_{worst} compared to ours, Table II reveals substantial increases in computation and communication delays due to the assignment of dense linear combinations of submatrices to the edge devices.

Coefficient determination time: Next, Table III shows a comparative analysis of various methods with respect to the time required for performing 10 trials to obtain a "good" set of random coefficients that ensures numerical stability of the system. As explained in Section IV-D, the techniques proposed in [36] and [29] involve partitioning matrix $\mathbf{A}$ into $\Delta_A = \text{LCM}(n, k_A)$ block-columns. For instance, when $n = 42$ and $s = 6$, $\Delta_A = 42$ is significantly larger than $k_A = 6$, which denotes the partition level in our approach. Consequently, when dealing with higher-sized matrices to determine the condition number, the methods proposed in [29] and [36] necessitate considerably more time compared to our approach.

Remark 5. We also conduct numerical simulations on matrix-vector multiplication (Sec. V in [1]), yielding results that mirror a similar trend observed in the matrix-matrix scenario.

Encoding weights: We also compare our approach more closely against the recent method developed in [31], in terms of encoding weights for different scenarios. The results are shown in Fig. 5. First, we consider a system of $n = 30$ devices and $s = 9$ stragglers within the matrix-vector scenario. Our proposed approach always matches the theoretical lower bound on the weight developed in Prop. 1 in the matrix-vector case, and outperforms the approach in [31].

Next we consider the matrix-matrix scenario, for two different systems: (a) $n = 36$ and $s = 8$ and (b) $n = 56$ and $s = 14$. While our method involves slightly higher weights than the theoretical lower bound for system (a), it matches the bound for system (b), and outperforms the approach in [31] in both cases. Here, the theoretical lower bound in system (a) is $\left\lceil \frac{(36-8)(8+1)}{36} \right\rceil = 7$, which is a prime number. Thus, because of the divisibility issue with $\omega_A, \omega_B > 1$, our proposed approach involves a slightly higher weight, $\omega = \omega_A \omega_B = 8$. Note that the dense coded approaches [25], [32], [33] involve a weight $n - s$, which is significantly higher than our proposed one and the method in [31], and hence, they are not included in the comparison in Fig. 5.

Numerical robustness and scalability: Finally, we consider the case of matrix-vector multiplication in different system architectures having different number of edge devices (n) and stragglers (s). The κ_{worst} values for different scenarios obtained from different available approaches are demonstrated in Fig. 6. These results confirm that our proposed approach consistently leads to smaller κ_{worst} values compared to the

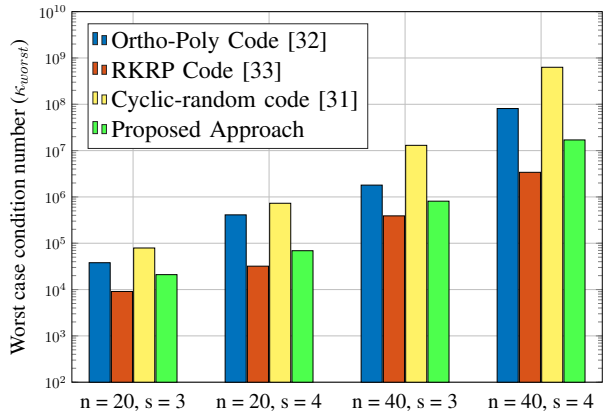

Fig. 6: Comparison of worst case condition numbers (κ_{worst}) of the encoding matrices obtained by different methods in distributed matrix-vector multiplication with different choices of n and s.

numerically stable approaches in [32] and [31], and provides competitive results compared to another numerically stable method in [33] for even larger values of n and s. This verifies the numerical robustness and the scalability of our proposed method for a larger network comprised of more edge devices. Note that in addition to the numerical robustness, unlike [33], our approach can leverage the inherent sparsity of the "input" matrices, and thus leads to reduced computation and communication delays, as demonstrated in Table II.

VII. Conclusion

In this study, we developed novel distributed matrix computation techniques to maintain the sparsity characteristics of input matrices while ensuring resilience against a maximum number of stragglers within specified storage constraints. First, we find a lower bound on the homogeneous weight of the encoded submatrices, and we show that our approaches meet that lower bound. Unlike conventional dense coded methods [23], [25], [32], [33], our proposed approach restricts coding within submatrices, thereby preserving the inherent sparsity of the input matrices $\mathbf{A}$ and $\mathbf{B}$ to a significant extent. Consequently, both worker computation and communication delays are markedly reduced compared to different coding techniques. In addition, unlike the sparsely coded methods in [29], [36], we demonstrate the extension of the approach to the heterogeneous setting, where the edge devices are rated with different computational and communication speeds.

Future research avenues include exploring server-less architectures where edge devices collaborate directly, rather than relying on a edge server for matrix encoding [8]. Enhancing performance in heterogeneous settings [60] by allocating multiple jobs with varying weights as proposed in [61], [62], and devising schemes where knowledge about edge devices is limited prior to job assignment, can also be important directions for future investigation.

Appendix

A. Proof of Claim 4

Proof. As discussed in Sec. V-2, we rearrange the $\mathcal{M}_i$'s such that $\tilde{\delta}_0 \geq \tilde{\delta}_1 \geq \tilde{\delta}_2 \geq \cdots \geq \tilde{\delta}_{k_A-1}$ and rename the corresponding $\mathcal{M}_i$'s as $\tilde{\mathcal{M}}_i$'s so that $\tilde{\delta}_i$ devices have been chosen from $\tilde{\mathcal{M}}_i$. Next we denote the minimum number of participating unknowns in $\tilde{\mathcal{M}}_i$ by ρ_i when $0 \leq i \leq k_A - 1$. The trivial lower bound for ρ_i is *zero* when $k_A - \omega_A + 1 \leq i \leq k_A - 1$, hence, $\sum_{i=0}^{k_A-1} \rho_i \geq \sum_{i=0}^{k_A-\omega_A} \rho_i$. Thus, in order to prove the lemma, for $m_0 = \sum_{i=0}^{k_A-1} \tilde{\delta}_i$, we need to show that

$$\sum_{i=0}^{k_A-\omega_A} \rho_i \;\geq\; \min\left(m_0 + \omega - 1, k\right). \tag{6}$$

Now we consider the following cases for each of which we show that (6) is true. Note that the minimum value of $\tilde{\delta}_0 = 1$, since $m_0 \geq 1$.

Case 1: $1 \leq \tilde{\delta}_0 \leq k_B - \omega_B$. In this case, according to (4),

$$\rho_0 = \omega_A \times \min(\omega_B + \tilde{\delta}_0 - 1, k_B) = \omega_A(\omega_B + \tilde{\delta}_0 - 1). \tag{7}$$

Case 1(a): $\tilde{\delta}_{k_A-\omega_A+1} = 0$. In this scenario, $\tilde{\delta}_{k_A-\omega_A+1} = \cdots = \tilde{\delta}_{k_A-1} = 0$, hence $\sum_{i=k_A-\omega_A+1}^{k_A-1} \tilde{\delta}_i = 0$. Thus,

$$\begin{aligned}
\sum_{i=0}^{k_A-\omega_A} \rho_i \;&=\; \rho_0 + \sum_{i=1}^{k_A-\omega_A} \rho_i \\
&\geq \omega_A(\omega_B + \tilde{\delta}_0 - 1) + \sum_{i=1}^{k_A-\omega_A} \tilde{\delta}_i + \sum_{i=k_A-\omega_A+1}^{k_A-1} \tilde{\delta}_i \\
&= \omega_A\tilde{\delta}_0 + \omega_A(\omega_B - 1) + \sum_{i=1}^{k_A-1} \tilde{\delta}_i \\
&= (\omega_A - 1)\tilde{\delta}_0 + \tilde{\delta}_0 + \sum_{i=1}^{k_A-1} \tilde{\delta}_i + \omega_A(\omega_B - 1) \\
&\geq \sum_{i=0}^{k_A-1} \tilde{\delta}_i + \omega_A\omega_B - 1 \;=\; m_0 + \omega - 1\,;
\end{aligned} \tag{8}$$

since $\tilde{\delta}_0 \geq 1$. Thus, (6) is true in this scenario.

Case 1(b): $\tilde{\delta}_{k_A-\omega_A+1} \geq 1$. In this scenario, $\tilde{\delta}_{k_A-\omega_A+1} \geq \tilde{\delta}_{k_A-\omega_A+2} \geq \cdots \geq \tilde{\delta}_{k_A-1}$, hence $\sum_{i=k_A-\omega_A+1}^{k_A-1} \tilde{\delta}_i \leq (\omega_A - 1)\tilde{\delta}_0$. Next note that according to (5), we have $\rho_i \geq \omega_B + \tilde{\delta}_i - 1$ for $1 \leq i \leq k_A - \omega_A$. Thus, we can say

$$\begin{aligned}
\sum_{i=0}^{k_A-\omega_A} \rho_i \;&=\; \rho_0 + \sum_{i=1}^{k_A-\omega_A} \rho_i \\
&\geq \omega_A(\omega_B + \tilde{\delta}_0 - 1) + \sum_{i=1}^{k_A-\omega_A} \left(\omega_B + \tilde{\delta}_i - 1\right) \\
&= \tilde{\delta}_0 + (\omega_A - 1)\tilde{\delta}_0 + \omega_A(\omega_B - 1) \\
&\quad + \sum_{i=1}^{k_A-\omega_A} \tilde{\delta}_i + (k_A - \omega_A)(\omega_B - 1) \\
&\geq \sum_{i=0}^{k_A-1} \tilde{\delta}_i + (k_A - \omega_A)(\omega_B - 1) + \omega_A(\omega_B - 1) \\
&\geq \sum_{i=0}^{k_A-1} \tilde{\delta}_i + (\omega_B - 1) + \omega_A(\omega_B - 1) \\
&\geq \sum_{i=0}^{k_A-1} \tilde{\delta}_i + \omega_A\omega_B - 1 \;=\; m_0 + \omega - 1\,;
\end{aligned} \tag{9}$$

as we assume $\omega_A \leq \omega_B$. Thus, (6) is true in this scenario too.

Case 2: $k_B - \omega_B < \tilde{\delta}_0 \leq k_B$. In this case, according to (4), we have $\rho_0 = \omega_A \times \min(\omega_B + \tilde{\delta}_0 - 1, k_B) = \omega_A k_B$.

Case 2(a): $\tilde{\delta}_{k_A-\omega_A} > k_B - \omega_B$. In this scenario, $\tilde{\delta}_0 \geq \tilde{\delta}_1 \geq \cdots \geq \tilde{\delta}_{k_A-\omega_A} => k_B - \omega_B$. Thus,

$$\begin{aligned}\sum_{i=0}^{k_A-\omega_A} \rho_i &= \rho_0 + \sum_{i=1}^{k_A-\omega_A} \rho_i \\ &= \omega_A k_B + (k_A - \omega_A)k_B = k_A k_B = k. \end{aligned} \tag{10}$$

where $\rho_i = k_B$ according to (5), for $i = 1, 2, \ldots, k_A - \omega_A$. Thus, (6) is true in this scenario.

Case 2(b): $\tilde{\delta}_{k_A-\omega_A} \leq k_B - \omega_B$. Thus, according to (5), $\rho_{k_A-\omega_A} = \tilde{\delta}_{k_A-\omega_A} + \omega_B - 1$. Moreover, since in this scenario, we have $\tilde{\delta}_{k_A-1} \leq \tilde{\delta}_{k_A-2} \leq \cdots \leq \tilde{\delta}_{k_A-\omega_A+1} \leq k_B - \omega_B$; hence we have $\sum_{i=k_A-\omega_A+1}^{k_A-1} \tilde{\delta}_i \leq (k_B - \omega_B)(\omega_A - 1)$. Now,

$$\begin{aligned}\sum_{i=0}^{k_A-\omega_A} \rho_i &= \rho_0 + \sum_{i=1}^{k_A-\omega_A-1} \rho_i + \rho_{k_A-\omega_A} \\ &\geq \omega_A k_B + \sum_{i=1}^{k_A-\omega_A-1} \tilde{\delta}_i + \left(\tilde{\delta}_{k_A-\omega_A} + \omega_B - 1\right) \\ &= (\omega_A - 1)k_B + k_B + \omega_B - 1 + \sum_{i=1}^{k_A-\omega_A} \tilde{\delta}_i \\ &\geq (\omega_A - 1)k_B + \omega_B - 1 - (k_B - \omega_B)(\omega_A - 1) \\ &\quad + \tilde{\delta}_0 + \sum_{i=1}^{k_A-\omega_A} \tilde{\delta}_i + \sum_{i=k_A-\omega_A+1}^{k_A-1} \tilde{\delta}_i \\ &= \omega_A k_B - k_B + \omega_B - 1 - \omega_A k_B + \omega_A \omega_B \\ &\quad + k_B - \omega_B + \sum_{i=1}^{k_A-1} \tilde{\delta}_i \\ &\geq \sum_{i=1}^{k_A-1} \tilde{\delta}_i + \omega_A \omega_B - 1 \,. \end{aligned} \tag{11}$$

Thus, (6) is true in this scenario; this concludes the proof. ■

B. Proof of Claim 5

Proof. Consider the edge devices in $\mathcal{W}_1$. According to Alg. 2, we assign a linear combination of $\mathbf{A}_{\ell\omega_A}, \mathbf{A}_{\ell\omega_A+1}, \ldots, \mathbf{A}_{(\ell+1)\omega_A-1}$ (indices modulo k_A) to edge device W_i, for $i = k, k+1, \ldots, n-1$ where $\ell = \text{mod}(i, k_A)$. In addition, we assign another linear combination of $\mathbf{B}_{m\omega_B}, \mathbf{B}_{m\omega_B+1}, \ldots, \mathbf{B}_{(m+1)\omega_B-1}$ (indices mod k_B) to device W_i, for $i = k, k+1, \ldots, n-1$, where $m = \left\lfloor \frac{i\,\omega_A}{k_A} \right\rfloor$.

Thus, the participating unknowns in edge device W_{k_A} are $\mathbf{A}_0^T\mathbf{B}_0, \ldots, \mathbf{A}_0^T\mathbf{B}_{\omega_B-1}, \mathbf{A}_1^T\mathbf{B}_0, \ldots, \ldots, \mathbf{A}_{\omega_A-1}^T\mathbf{B}_{\omega_B-1}$. Similarly, the participating submatrices in W_{k_A+1} are $\mathbf{A}_{\omega_A}^T\mathbf{B}_0, \ldots, \mathbf{A}_{\omega_A}^T\mathbf{B}_{\omega_B-1}, \mathbf{A}_{\omega_A+1}^T\mathbf{B}_0, \ldots, \ldots, \mathbf{A}_{2\omega_A-1}^T\mathbf{B}_{\omega_B-1}$. In a consequence, $\omega = \omega_A\omega_B$ number of submatrices participate in each of those s edge devices in a cyclic fashion.

Now, denote the number of appearances of any submatrix $\mathbf{A}_i^T\mathbf{B}_j$ within the devices in $\mathcal{W}_1$ by $\mathbf{v}_{ij} \geq 0$. Thus, for any $0 \leq i, p \leq k_A - 1$ and $0 \leq j, q \leq k_B - 1$, we have $|\mathbf{v}_{ij} - \mathbf{v}_{pq}| \leq 1$, where $\sum_{i=0}^{k_A-1}\sum_{j=0}^{k_B-1} \mathbf{v}_{ij} = s\omega$. Thus, the average of these $\mathbf{v}_{ij}$'s is $\kappa = \frac{s\omega}{k}$. Since for every pair of (i, j) and (p, q), we have $|\mathbf{v}_{ij} - \mathbf{v}_{pq}| \leq 1$, thus, we can say that $\mathbf{v}_{ij} \geq \lfloor\kappa\rfloor = \left\lfloor \frac{s\omega}{k} \right\rfloor$. It indicates that within all s devices of $\mathcal{W}_1$, every unknown participates in at least $\lfloor\kappa\rfloor$ times over $\lfloor\kappa\rfloor$ distinct devices. In other words, any unknown may not participate in at most $s - \lfloor\kappa\rfloor$ devices within the devices of $\mathcal{W}_1$.

First, consider the case, $k = s$. Here, every unknown participates in $\lfloor\kappa\rfloor = \omega$ devices, therefore, any unknown does not participate in $s - \omega$ devices. But, we choose any $m_1 \geq \omega$ devices in $\mathcal{W}_1$, where $\omega = \lceil \frac{s+1}{2} \rceil$, since $k = s$. Thus,

$$2\omega \geq s + 1 > s \;\text{ which indicates that, }\; \omega > s - \omega.$$

In addition, since $m_1 \geq \omega$, we claim that $m_1 > s - \omega_A$. Thus, every submatrix will participate at least once within those chosen m_1 devices, hence $|\mathcal{X}_1| = k = k_A k_B$.

Next, consider the other case when $k > s$. Again, since we choose any arbitrary $m_1 \geq \omega$ devices in $\mathcal{W}_1$, we are leaving $s - m_1$ devices in $\mathcal{W}_1$. But

$$s - m_1 \leq s - \omega < s - \lfloor\kappa\rfloor.$$

The second inequality holds since $s < k$, hence, $\lfloor\kappa\rfloor < \omega$. Thus, every submatrix will participate at least once within those $m_1 \geq \omega$ devices, hence $|\mathcal{X}_1| = k$. ■

REFERENCES

[1] A. B. Das, A. Ramamoorthy, D. J. Love, and C. G. Brinton, "Preserving sparsity and privacy in straggler-resilient distributed matrix computations," in *Proc. of Annual Allerton Conf. Comm., Control, and Comput. (Allerton)*, 2023, pp. 1–8.

[2] E. Vedadi and H. Seferoglu, "Adaptive coding for matrix multiplication at edge networks," in *Proc. of IEEE Intl. Symp. on Info. Th. (ISIT)*, 2021, pp. 1064–1069.

[3] N. Van Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Joint coding and scheduling optimization for distributed learning over wireless edge networks," *IEEE J. Sel. Area. Comm.*, vol. 40, no. 2, pp. 484–498, 2022.

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

[5] S. Hosseinalipour, C. G. Brinton, V. Aggarwal, H. Dai, and M. Chiang, "From federated to fog learning: Distributed machine learning over heterogeneous wireless networks," *IEEE Commun. Mag.*, vol. 58, no. 12, pp. 41–47, 2020.

[6] M. Amadeo, C. Campolo, A. Molinaro, G. Ruggeri, and G. Singh, "Mitigating the communication straggler effect in federated learning via named data networking," *IEEE Comm. Mag.*, pp. 1–7, 2024.

[7] S. Kadhe, O. O. Koyluoglu, and K. Ramchandran, "Communication-efficient gradient coding for straggler mitigation in distributed learning," in *Proc. of IEEE Intl. Symp. on Info. Th. (ISIT)*, 2020, pp. 2634–2639.

[8] S. Prakash, S. Dhakal, M. R. Akdeniz, Y. Yona, S. Talwar, S. Avestimehr, and N. Himayat, "Coded computing for low-latency federated learning over wireless edge networks," *IEEE J. Sel. Area. Comm.*, vol. 39, no. 1, pp. 233–250, 2021.

[9] S. Dhakal, S. Prakash, Y. Yona, S. Talwar, and N. Himayat, "Coded federated learning," in *IEEE Globecom Workshops*, 2019, pp. 1–6.

[10] A. B. Das, A. Ramamoorthy, D. J. Love, and C. G. Brinton, "Coded matrix computations for D2D-enabled linearized federated learning," in *Proc. of IEEE Intl. Conf. on Acoustics, Speech and Sig. Proc. (ICASSP)*, 2023, pp. 1–5.

[11] A. Asheralieva and D. Niyato, "Throughput-efficient lagrange coded private blockchain for secured iot systems," *IEEE Internet of Things Jour.*, vol. 8, no. 19, pp. 14 874–14 895, 2021.

[12] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Low-latency federated learning and blockchain for edge association in digital twin empowered 6G networks," *IEEE Trans. Ind. Inform.*, vol. 17, no. 7, pp. 5098–5107, 2021.

[13] X. Ma and D. Xu, "Torr: A lightweight blockchain for decentralized federated learning," *IEEE Internet of Things Jour.*, vol. 11, no. 1, pp. 1028–1040, 2024.

[14] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely coded computing," in *Proc. of IEEE Intl. Symp. on Info. Th. (ISIT)*, 2019, pp. 2798–2802.

[15] B. Buyukates and S. Ulukus, "Timely distributed computation with stragglers," *IEEE Trans. Commun.*, vol. 68, no. 9, pp. 5273–5282, 2020.

[16] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," *Proc. of Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017.

[17] D. Data, L. Song, and S. N. Diggavi, "Data encoding for byzantine-resilient distributed optimization," *IEEE Trans. Info. Th.*, vol. 67, no. 2, pp. 1117–1140, 2021.

[18] C. Karakus, Y. Sun, and S. Diggavi, "Encoded distributed optimization," in *Proc. of IEEE Intl. Symp. on Info. Th. (ISIT)*, 2017, pp. 2890–2894.

[19] J. Zhu, Y. Pu, V. Gupta, C. Tomlin, and K. Ramchandran, "A sequential approximation framework for coded distributed optimization," in *Proc. of Annual Allerton Conf. Comm., Control, and Comput. (Allerton)*, 2017, pp. 1240–1247.

[20] U. Demirbaga and G. S. Aujla, "Mapchain: A blockchain-based verifiable healthcare service management in iot-based big data ecosystem," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 4, pp. 3896–3907, 2022.

[21] A. Gupta, S. Misra, N. Pathak, and D. Das, "Fedcare: Federated learning for resource-constrained healthcare devices in iomt system," *IEEE Trans. Comput. Soc.*, vol. 10, no. 4, pp. 1587–1596, 2023.

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

[23] A. B. Das, A. Ramamoorthy, and N. Vaswani, "Efficient and robust distributed matrix computations via convolutional coding," *IEEE Trans. Info. Th.*, vol. 67, no. 9, pp. 6266–6282, 2021.

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

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

[26] A. B. Das, L. Tang, and A. Ramamoorthy, "C^3LES : Codes for coded computation that leverage stragglers," in *Proc. of IEEE Info. Th. Workshop (ITW)*, 2018, pp. 1–5.

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

[28] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proc. of Intl. Conf. on Mach. Learn. (ICML)*, 2017, pp. 3368–3376.

[29] A. B. Das and A. Ramamoorthy, "A unified treatment of partial stragglers and sparse matrices in coded matrix computation," *IEEE Jour. on Sel. Area. in Info. Th.*, vol. 3, no. 2, pp. 241–256, 2022.

[30] A. K. Pradhan, A. Heidarzadeh, and K. R. Narayanan, "Factored LT and factored raptor codes for large-scale distributed matrix multiplication," *IEEE Jour. Sel. Area. Info. Th.*, vol. 2, no. 3, pp. 893–906, 2021.

[31] A. B. Das, A. Ramamoorthy, D. J. Love, and C. G. Brinton, "Distributed matrix computations with low-weight encodings," *IEEE Jour. Sel. Area. Info. Th.*, vol. 4, pp. 363–378, 2023.

[32] M. Fahim and V. R. Cadambe, "Numerically stable polynomially coded computing," *IEEE Trans. Info. Th.*, vol. 67, no. 5, pp. 2758–2785, 2021.

[33] A. M. Subramaniam, A. Heidarzadeh, and K. R. Narayanan, "Random Khatri-Rao-product codes for numerically-stable distributed matrix multiplication," in *Proc. of Annual Allerton Conf. Comm., Control, and Comput. (Allerton)*, Sep. 2019, pp. 253–259.

[34] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *Proc. of Intl. Conf. on Mach. Learn. (ICML)*, 2018, pp. 5152—5160.

[35] M. Xhemrishi, R. Bitar, and A. Wachter-Zeh, "Distributed matrix-vector multiplication with sparsity and privacy guarantees," in *Proc. of IEEE Intl. Symp. on Info. Th. (ISIT)*, 2022, pp. 1028–1033.

[36] A. B. Das and A. Ramamoorthy, "Coded sparse matrix computation schemes that leverage partial stragglers," *IEEE Trans. Info. Th.*, vol. 68, no. 6, pp. 4156–4181, 2022.

[37] J. Li and C. Hollanti, "Private and secure distributed matrix multiplication schemes for replicated or mds-coded servers," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 659–669, 2022.

[38] M. Aliasgari, O. Simeone, and J. Kliewer, "Private and secure distributed matrix multiplication with flexible communication load," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 2722–2734, 2020.

[39] W.-T. Chang and R. Tandon, "On the capacity of secure distributed matrix multiplication," in *Proc. of IEEE Glob. Comm. Conf. (GLOBECOM)*, 2018, pp. 1–6.

[40] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *ACM on Meas. and Analysis of Comp. Syst.*, vol. 3, no. 3, pp. 1–40, 2019.

[41] A. B. Das and A. Ramamoorthy, "Distributed matrix-vector multiplication: A convolutional coding approach," in *Proc. of IEEE Intl. Symp. on Info. Th. (ISIT)*, 2019, pp. 3022–3026.

[42] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, "Dynamic heterogeneity-aware coded cooperative computation at the edge," in *Proc. of IEEE Intl. Conf. on Network Protocols (ICNP)*, 2018, pp. 23–33.

[43] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city internet of things," *IEEE Internet of Things Jour.*, vol. 7, no. 9, pp. 8099–8110, 2020.

[44] H. Habibzadeh, Z. Qin, T. Soyata, and B. Kantarci, "Large-scale distributed dedicated- and non-dedicated smart city sensing systems," *IEEE Sensors Jour.*, vol. 17, no. 23, pp. 7649–7658, 2017.

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

[46] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. of IEEE Intl. Symp. on Info. Th. (ISIT)*, 2017, pp. 2418–2422.

[47] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate polynomial coding for efficient distributed matrix multiplication," *IEEE Jour. Sel. Area. Info. Th.*, vol. 2, no. 3, pp. 814–829, 2021.

[48] A. Ramamoorthy and L. Tang, "Numerically stable coded matrix computations via circulant and rotation matrix embeddings," *IEEE Trans. Info. Th.*, vol. 68, no. 4, pp. 2684–2703, 2022.

[49] SuiteSparse Matrix Collection. [Online]. Available: https://sparse.tamu.edu/

[50] H. Yao, W. Yu, and X. Wang, "A feature memory rearrangement network for visual inspection of textured surface defects toward edge intelligent manufacturing," *IEEE Trans. Autom. Sci. Eng.*, vol. 20, no. 4, pp. 2616–2635, 2023.

[51] W. Yu, Y. Liu, T. Dillon, and W. Rahayu, "Edge computing-assisted iot framework with an autoencoder for fault detection in manufacturing predictive maintenance," *IEEE Trans. Ind. Inform.*, vol. 19, no. 4, pp. 5701–5710, 2023.

[52] R. G. Lins and S. N. Givigi, "Cooperative robotics and machine learning for smart manufacturing: Platform design and trends within the context of industrial internet of things," *IEEE Access*, vol. 9, pp. 95 444–95 455, 2021.

[53] E. Wang, M. Zhang, X. Cheng, Y. Yang, W. Liu, H. Yu, L. Wang, and J. Zhang, "Deep learning-enabled sparse industrial crowdsensing and prediction," *IEEE Trans. Ind. Inform.*, vol. 17, no. 9, pp. 6170–6181, 2021.

[54] R. Ji, A. Heidarzadeh, and K. R. Narayanan, "Sparse random khatri-rao product codes for distributed matrix multiplication," in *Proc. of IEEE Info. Th. Workshop (ITW)*, 2022, pp. 416–421.

[55] J. Marshall. Hall, *Combinatorial theory*. Wiley, 1986.

[56] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities," *Jour. of the ACM*, vol. 27, no. 4, pp. 701–717, 1980.

[57] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (PCA)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303–342, 1993.

[58] P. Xanthopoulos, P. M. Pardalos, T. B. Trafalis, P. Xanthopoulos, P. M. Pardalos, and T. B. Trafalis, "Linear discriminant analysis," *Robust data mining*, pp. 27–33, 2013.

[59] R. Ji, A. Heidarzadeh, and K. R. Narayanan, "Sparse random khatri-rao product codes for distributed matrix multiplication," in *Proc. of IEEE Info. Th. Workshop (ITW)*, 2022, pp. 416–421.

[60] W. Xu, Z. Yang, D. W. K. Ng, M. Levorato, Y. C. Eldar, and M. Debbah, "Edge learning for b5g networks with distributed signal processing: Semantic communication, edge computing, and wireless sensing," *IEEE Jour. Sel. Topics in Sig. Proc.*, vol. 17, no. 1, pp. 9–39, 2023.

[61] E. Ozfatura, S. Ulukus, and D. Gündüz, "Coded distributed computing with partial recovery," *IEEE Trans. Info. Th.*, vol. 68, no. 3, pp. 1945–1959, 2021.

[62] E. Ozfatura, B. Buyukates, D. Gündüz, and S. Ulukus, "Age-based coded computation for bias reduction in distributed learning," in *Proc. of IEEE Glob. Comm. Conf. (GLOBECOM)*, 2020, pp. 1–6.