This is the peer reviewd version of the followng article:

Coded Distributed Computing with Partial Recovery / Ozfatura, E., Ulukus, S., Gunduz, D.. - In: IEEE TRANSACTIONS ON INFORMATION THEORY. - ISSN 0018-9448. - 68:3(2022), pp. 1945-1959. [10.1109/TIT.2021.3133791]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

17/11/2026 18:03

(Article begins on next page)

Coded Distributed Computing with Partial Recovery

Emre Ozfatura, Sennur Ulukus, and Deniz Gündüz

arXiv:2007.02191v2 [cs.IT] 6 Dec 2021

***Abstract*—Coded computation techniques provide robustness against *straggling* workers in distributed computing. However, most of the existing schemes require exact provisioning of the straggling behavior and ignore the computations carried out by straggling workers. Moreover, these schemes are typically designed to recover the desired computation results accurately, while in many machine learning and iterative optimization algorithms, faster approximate solutions are known to result in an improvement in the overall convergence time. In this paper, we first introduce a novel coded matrix-vector multiplication scheme, called *coded computation with partial recovery (CCPR)*, which benefits from the advantages of both coded and uncoded computation schemes, and reduces both the computation time and the decoding complexity by allowing a trade-off between the accuracy and the speed of computation. We then extend this approach to distributed implementation of more general computation tasks by proposing a coded communication scheme with partial recovery, where the results of subtasks computed by the workers are coded before being communicated. Numerical simulations on a large linear regression task confirm the benefits of the proposed scheme in terms of the trade-off between the computation accuracy and latency.**

***Index Terms*—Coded computation, distributed computation, maximum distance separable (MDS) code, linear codes, rateless codes, stragglers.**

I. Introduction

One of the key enablers of efficient machine learning solutions is the availability of large datasets. However, the ever growing size of the datasets and the complexity of the models trained on them lead also to an increase in the computational complexity and storage requirements of the algorithms employed. In parallel, there is a growing availability of cloud computing platforms (such as Amazon Web Services, Microsoft Azure and Google Cloud Functions) that offer computational resources to users to carry out demanding computation tasks. The associated distributed computation framework allows harnessing the computation and memory resources of multiple heterogeneous computation servers, referred to as *workers*.

In the most common implementation of distributed computation, a *parameter server (PS)* divides the main computational task into several subtasks and assigns them to workers. Each worker executes the computation tasks assigned to it, and conveys the result to the PS. Having received the results from all the workers, the PS combines them to obtain the result of the main computation task. In principle, such a distributed computation framework should achieve a speed-up factor proportional to the number of workers employed. However, in real implementations, the overall computation time is constrained by the slowest, so-called *straggling worker(s)*. Moreover, as the number of employed workers increases, communication starts to become more complex and to introduce additional delays, which can aggravate the straggler problem. To remedy the delays due to straggling workers, various straggler-tolerant distributed computation schemes have been introduced recently, which build upon the idea of assigning redundant computations/subtasks to workers, to let faster workers compensate for the stragglers [1]–[42].

A. Motivation

We will motivate the proposed distributed computation framework on a simple regression problem. In linear regression, the goal is to minimize the empirical mean squared-error:

$$L(\boldsymbol{\theta}) \triangleq \frac{1}{2N}\sum_{i=1}^{N}(y_i - \mathbf{x}_i^T\boldsymbol{\theta})^2, \tag{1}$$

where $\mathbf{x}_1,\dots,\mathbf{x}_N \in \mathbb{R}^d$ are the data points with corresponding labels $y_1,\dots,y_N \in \mathbb{R}$, and $\boldsymbol{\theta} \in \mathbb{R}^d$ is the parameter vector. The optimal parameter vector can be obtained iteratively by gradient descent (GD), in which the parameter vector is updated iteratively as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t), \tag{2}$$

where η_t is the learning rate at the t-th iteration. Gradient of the loss function in (1) can be written as

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_t) = \mathbf{X}^T\mathbf{X}\boldsymbol{\theta}_t - \mathbf{X}^T\mathbf{y}, \tag{3}$$

where $\mathbf{X} = [\mathbf{x}_1,\dots,\mathbf{x}_N]^T$ and $\mathbf{y} = [y_1,\dots,y_N]^T$. In the gradient expression, only $\boldsymbol{\theta}_t$ changes over iterations; hence, the key computational task at each iteration is the matrix-vector multiplication $\mathbf{W}\boldsymbol{\theta}_t$, where $\mathbf{W} \triangleq \mathbf{X}^T\mathbf{X} \in \mathbb{R}^{d\times d}$.

B. Coded Distributed Matrix-Vector Multiplication

The execution of $\mathbf{W}\boldsymbol{\theta}_t$ can be distributed across K *workers* by simply dividing $\mathbf{W}$ row-wise into K disjoint submatrices and assigning each submatrix to one of the workers. However, the computation time of this naive approach will be limited by the *straggling* worker(s). The main challenge in this setup arises because the straggling behaviour (due either to the computation speed of the workers or the delays in communication) varies over time, and its realization at each iteration is not

This paper was presented in part at the 2019 IEEE International Conference on Acoustics, Speech and Signal Processing in Brighton, UK.

Emre Ozfatura and Deniz Gündüz are with Information Processing and Communications Lab, Department of Electrical and Electronic Engineering, Imperial College London Email: {m.ozfatura, d.gunduz} @imperial.ac.uk.

Sennur Ulukus is with the Department of Electrical and Computer Engineering, University of Maryland.

This work was supported in part by the Marie Sklodowska-Curie Action SCAVENGE (grant agreement no. 675891), by the European Research Council (ERC) Starting Grant BEACON (grant agreement no. 677854), and UK EPSRC (EP/T023600/1) under the CHIST-ERA program (CHISTERA-18-SDCDN-001).

known in advance. On the other hand, statistical knowledge about the computation and communication latency for each worker can be acquired over time, and used for a more efficient allocation of computation tasks (e.g. as in [28], [35], [36]) as well as the coding scheme employed. For the sake of simplicity we assume homogeneous workers in this work.

Coded computation has been introduced to tolerate stragglers in matrix-vector multiplication by encoding the $\mathbf{W}$ matrix, and distributing the partitions of this encoded matrix among the workers, to achieve redundancy [15]–[27]. One well-known method to introduce redundancy in matrix-vector multiplication is to utilize maximum distance separable (MDS) codes to encode $\mathbf{W}$ [15]. To elucidate the MDS-coded computation (MCC) we can divide $\mathbf{W}$ into $\bar{K}$ disjoint submatrices, $\mathbf{W}_1, \dots, \mathbf{W}_{\bar{K}} \in \mathbb{R}^{\bar{d} \times d}$, $\bar{d} = d/\bar{K}$, which are then encoded with a $(\bar{K}, K)$ MDS code. Each coded submatrix is assigned to a different worker, which multiplies it with $\boldsymbol{\theta}_t$, and returns the result to the PS. The PS can recover $\mathbf{W}\boldsymbol{\theta}_t$ from the results of any $\bar{K}$ workers.

Note that, up to $K - \bar{K}$ stragglers can be tolerated with MCC at the expense of increasing the *computation load* of each worker by $r = K/\bar{K}$; that is, each worker is assigned r times more computations compared to the naive approach of equally dividing all the required computations among the workers. Alternative to MDS codes [15], [16], [22], LDPC codes [17], and rateless codes [23] have also been studied for straggler-tolerant coded computation in the literature.

C. Computation-communication trade-off

Conventional straggler-aware designs assume that a single message is transmitted by each worker after completing its assigned computation task. Under this limitation, straggler-aware schemes require exact provisioning of the straggler behaviour, and otherwise, suffer from *over-computation* and *under-utilization* [43]. To overcome these obstacles one can allow each worker to send multiple messages to the PS at each iteration, which we refer to as *multi-message communication (MMC)* [1], [3], [13], [16], [19], [23], [24], [38]. However, MMC may introduce additional delays due to the communication overhead. Hence, with MMC the objective is to find an optimal operating point that balances the computation and communication latencies [43]. One recent approach for coded matrix-vector multiplication with MMC is to utilize rateless codes [23] due to their advantages of better utilizing the computational resources and low decoding complexity at the PS. However, in practice, rateless codes reach the target coding rates only if the number of coded messages are sufficiently large, which may not be desired in distributed computation framework since it leads to a congestion at the PS. Hence, the design of a code structure for distributed computation that can reduce the computation time without inducing an overwhelming communication overhead is an open challenge that we address in this paper.

D. Computation accuracy-computation speed trade-off

In the case of iterative optimization algorithms we can consider the accuracy of the computations at each iteration as another dimension of the trade-off governing the overall convergence behaviour. For example, when applying gradient descent over large datasets, computation of the gradient in (3) at each iteration can become very costly. The most common alternative iterative optimization framework for large scale learning problems is *stochastic gradient descent (SGD)*, which uses an estimate of the gradient in (3) at each iteration, evaluated on a random subset of the dataset. Hence, by changing the size of the sampled dataset it is possible to seek a balance between the accuracy of the gradient estimate and the computation time. This trade-off is actually employed in several practical implementations [44], [45].

On the other hand, a vast majority of the coded computation schemes in the literature are designed for full gradient recovery. Nevertheless, a simple uncoded computation scheme with MMC [3], [19] can exploit partial computations performed by straggling workers, while also providing the PS a certain flexibility to terminate an iteration when a sufficient number of computations are received. Accordingly, our goal here is to design a coded computing framework that can efficiently benefit from redundant computations with the flexibility of partial gradient computations. To this end, we introduce a novel hybrid scheme for distributed matrix-vector multiplication, called *coded computation with partial recovery* (CCPR), bringing together the advantages of uncoded computation, such as low decoding complexity and partial gradient updates, with those of coded computation, such as reduced per-iteration completion time and reduced communication load.

We also want to highlight that, in most of the coded computation schemes in the literature, encoding is executed in PS in a centralized manner, and coded submatrices are distributed to workers; however, in our proposed strategy, as explained in Section IV, encoding step can be executed in a decentralized manner. Such local encoding provides two key advantages; first, it is possible to dynamically change the codewords over time based on the realization of the straggler behaviour [46], which is particularly desired when the straggler behavior is correlated over time; second, as further explained in Section V, it allows us to extend the CCPR strategy to more general distributed learning problems.

To the best of our knowledge, the partial recovery approach was first introduced in our preliminary work [26]. In this paper, we extend this approach and provide a more comprehensive analysis. Our specific contributions can be summarized as follows:

- We provide a general framework, and highlight certain design principles to efficiently employ partial recovery in a coded computation scenario, particularly with MMC.
- Based on these design principles, we introduce *random circularly shifted (RCS) codes* for distributed matrix-vector multiplication.
- We provide a generalization of RCS codes to the distributed implementation of computation tasks beyond matrix-vector multiplication by proposing a gradient coding scheme with partial computation.
- Through numerical experiments in a linear regression problem, we show that RCS codes outperform existing distributed learning schemes across straggling workers,

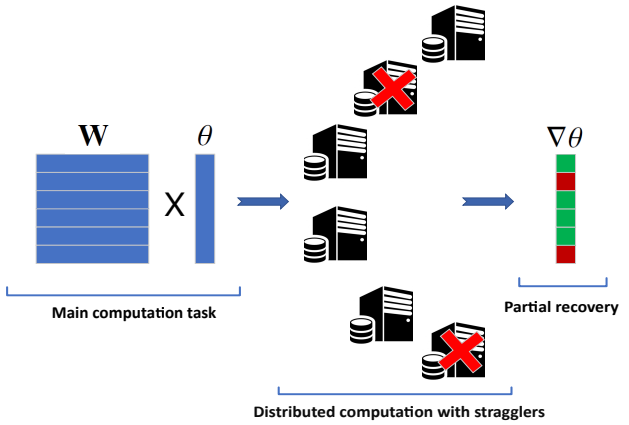


Fig. 1: Illustration of partial recovery in a naive distributed computation scenario with 6 workers, 2 of which are stragglers.

and also present the trade-offs between the update accuracy, communication latency and computation time achieved by these codes.

II. Coded computation with partial recovery

In conventional coded computation schemes, the PS waits until a sufficient number of computations are gathered from the workers to recover the correct results of the underlying computation task. In contrast, the partial recovery strategy does not necessarily aim at recovering the results accurately. In particular, for the matrix-vector multiplication task of $\mathbf{W}\boldsymbol{\theta}$, $\mathbf{W} \in \mathbb{R}^{d\times d}$, $\boldsymbol{\theta} \in \mathbb{R}^{d}$, we will target recovering only a subset of the entries of the d-dimensional result vector. We define the percentage of entries of $\mathbf{W}\boldsymbol{\theta}$ to be recovered as the *tolerance*, which will be dictated by the underlying computation task.

For encoding, we utilize a general linear code structure. Matrix $\mathbf{W}$ is initially divided into K disjoint submatrices $\mathbf{W}_1, \dots, \mathbf{W}_K \in \mathbb{R}^{d/K\times d}$, where we assume K divides d, as otherwise this can be satisfied with zero padding. Then, r coded submatrices, $\widetilde{\mathbf{W}}_{i,1}, \dots, \widetilde{\mathbf{W}}_{i,r}$, are assigned to each worker i for computation, where each coded matrix $\widetilde{\mathbf{W}}_{i,j}$ is a linear combination of K submatrices, i.e.,

$$\widetilde{\mathbf{W}}_{i,j} = \sum_{k\in[K]} \alpha^{(i)}_{j,k}\mathbf{W}_k. \tag{4}$$

Following the initial encoding phase, the ith worker performs the computations $\widetilde{\mathbf{W}}_{i,1}\boldsymbol{\theta}, \dots, \widetilde{\mathbf{W}}_{i,r}\boldsymbol{\theta}$ in the given order and sends the result as soon as it is completed. We remark that, in the considered MMC scenario, the order of the assigned coded computations affects the completion time; therefore, we introduce the *computation assignment matrix* $\mathbf{C}$ to represent a coded computation strategy, i.e,

$$\mathbf{C} \triangleq \begin{bmatrix} \widetilde{\mathbf{W}}_{1,1} & \widetilde{\mathbf{W}}_{2,1} & \dots & \widetilde{\mathbf{W}}_{K,1} \\ \widetilde{\mathbf{W}}_{1,2} & \widetilde{\mathbf{W}}_{2,2} & \dots & \widetilde{\mathbf{W}}_{K,2} \\ \vdots & \vdots & \dots & \vdots \\ \widetilde{\mathbf{W}}_{1,r} & \widetilde{\mathbf{W}}_{2,r} & \dots & \widetilde{\mathbf{W}}_{K,r} \end{bmatrix},$$

where, $\mathbf{C}$ indicates the execution order of the assigned computation tasks to each worker. More specifically, submatrix $\mathbf{C}(j,i)$, $j \in [r]$, $i \in [K]$, denotes the jth computation task to be executed by the ith worker.

When partial recovery is allowed, PS waits until $(1-q)\times 100$ percent of the entries of the result vector are successfully recovered. We call the parameter q as the *tolerance*, which is a design parameter.

Consider a given a distributed coded computation strategy for K workers and redundancy parameter r, denoted by $\Pi(K,r)$. Then, let $T_{\Pi(K,r)}(q)$ and $M_{\Pi(K,r)}(q)$ be the expected computation time and the expected number of received messages to recover $K(1-q)$ partial results, respectively. In the scope of this work, our goal is to design a coded computation strategy, for the scenario where the system operates at set of different tolerance levels $\boldsymbol{Q}$, and we want to achieve "acceptable performance", both in terms of $T_{\Pi(K,r)}(q)$ and $M_{\Pi(K,r)}(q)$, within the whole spectrum of q. Based on system requirements and the computational task one can introduce certain constraints on these terms and can prioritize certain tolerance level. Hence, we want to introduce a novel coded computation framework so that one can search for the suitable coding strategy, utilizing the same structure, according to given performance requirements. On top of that proposed framework has two more advantages having a low decoding complexity and its re-configurable nature which allows to update the coding strategy in parallel to a change in the performance requirements.

Let us first present a simple example to show how the proposed coded computation scheme with partial recovery can improve upon other schemes in the literature, such as MDS coding or uncoded computation with MMC (UC-MMC).

A. Motivating example

Consider $K = 4$ workers and assume that $\mathbf{W}$ is divided into 4 submatrices $\mathbf{W}_1, \dots, \mathbf{W}_4$. Let us first consider two known distributed computation schemes, namely UC-MM [3], [19] and MDS-coded computation (MCC) [15]. Each scheme is defined by its computation assignment matrix.

In MDS-coded computation, linearly independent coded computation tasks are distributed to the workers as follows:

$$\mathbf{C}_{MDS} = \left[\begin{bmatrix} \mathbf{W}_1 + \mathbf{W}_3 \\ \mathbf{W}_2 + \mathbf{W}_4 \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 + 2\mathbf{W}_3 \\ \mathbf{W}_2 + 2\mathbf{W}_4 \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 + 4\mathbf{W}_3 \\ \mathbf{W}_2 + 4\mathbf{W}_4 \end{bmatrix} \begin{bmatrix} \mathbf{W}_1 + 8\mathbf{W}_3 \\ \mathbf{W}_2 + 8\mathbf{W}_4 \end{bmatrix} \right].$$

$\mathbf{C}_{MDS}$ consists of a single row of computation tasks since each worker sends the results of its computations only after all of them are completed, e.g., first worker sends the concatenation of $[(\mathbf{W}_1+\mathbf{W}_3)\boldsymbol{\theta} \;\; (\mathbf{W}_2+\mathbf{W}_4)\boldsymbol{\theta}]$ after completing both computations. $\mathbf{C}_{MDS}$ above corresponds to a $(2,4)$ MDS code; hence, the PS can recover the full gradient from the results of any two workers.

In the UC-MMC scheme with cyclic shifted computation assignment [19], the computation scheduling matrix is given by

$$\mathbf{C}_{UC-MMC} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 & \mathbf{W}_1 \end{bmatrix},$$

and each worker sends the results of its computations sequentially, as soon as each of them is completed. This helps

Cumulative computation type	MCC $n_m(\mathbf{N}_i)$	UC-MMC $n_u(\mathbf{N}_i)$	CCPR $n_c(\mathbf{N}_i)$
$\mathbf{N}_1 : N_2 = 4, N_1 = 0, N_0 = 0$	1	1	1
$\mathbf{N}_2 : N_2 = 3, N_1 = 1, N_0 = 0$	4	4	4
$\mathbf{N}_3 : N_2 = 3, N_1 = 0, N_0 = 1$	4	4	4
$\mathbf{N}_4 : N_2 = 2, N_1 = 2, N_0 = 0$	6	6	6
$\mathbf{N}_5 : N_2 = 2, N_1 = 1, N_0 = 1$	12	8	12
$\mathbf{N}_6 : N_2 = 2, N_1 = 0, N_0 = 2$	6	2	6
$\mathbf{N}_7 : N_2 = 1, N_1 = 3, N_0 = 0$	0	4	4
$\mathbf{N}_8 : N_2 = 1, N_1 = 2, N_0 = 1$	0	4	8
$\mathbf{N}_9 : N_2 = 0, N_1 = 4, N_0 = 0$	0	1	1

TABLE I: Number of successful score vectors for each cumulative computation that can accurately recover the computation task with $K = 4$ and $r = 2$.

to reduce the per-iteration completion time with an increase in the communication load [3], [19]. With UC-MMC, full gradient can be recovered even if each worker performs only one computation, which is faster if the workers have similar speeds.

The computation scheduling matrix of the proposed CCPR scheme is given by

$$\mathbf{C}_{CCPR} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 + \mathbf{W}_4 & \mathbf{W}_1 + \mathbf{W}_3 & \mathbf{W}_2 + \mathbf{W}_4 & \mathbf{W}_1 + \mathbf{W}_2 \end{bmatrix},$$

which is a combination of the uncoded and coded approaches. Below we illustrate the advantages of this scheme by comparing its performance for both accurate and approximate computations. For this analysis we need to introduce a few definitions.

Let $N_s(t)$ denote the number of workers that have completed exactly s computations by time t, $s = 0, \dots, r$. We define $\mathbf{N}(t) \triangleq (N_r(t), \dots, N_0(t))$ as the *cumulative computation type* at time t. Additionally, we introduce the K-dimensional *score vector* $\mathbf{S}(t) = [s_1(t), \dots, s_K(t)]$, where $s_i(t)$ denotes the number of computations completed and communicated by the ith worker by time t. We will call a score vector *successful* if it allows the recovery of the desired computation task at the PS. We note that, due to the homogeneous worker assumption, the probability of experiencing any score vector with the same cumulative computation type is the same. Therefore, what is important for the overall computation time statistics is the number of successful score vectors corresponding to each computation type.

B. Full gradient performance

Let $n_m(\mathbf{N}), n_u(\mathbf{N})$ and $n_c(\mathbf{N})$ denote the number of distinct succesfull score vectors with the cumulative computation type $\mathbf{N}$ that allow the recovery of $\mathbf{W}\boldsymbol{\theta}$ for the MDS, UC-MMC, and CCPR schemes, respectively. For instance, for the cumulative computation type $\mathbf{N} = (1, 2, 1)$, MDS scheme cannot recover the full gradient; however, UC-MMC can recover the full gradient for four $\mathbf{S}$ vectors; $\mathbf{S} = [2, 0, 1, 1]$, $\mathbf{S} = [1, 2, 0, 1]$, $\mathbf{S} = [1, 1, 2, 0]$ and $\mathbf{S} = [0, 1, 1, 2]$, hence $n_u(\mathbf{N}) = 4$. Finally, in the CCPR scheme, there are in total $n_c(\mathbf{N}) = 8$ successful $\mathbf{S}$ vectors; $[2, 1, 0, 1]$, $[2, 1, 1, 0]$, $[1, 2, 0, 1]$, $[0, 2, 1, 1]$, $[1, 0, 2, 1]$, $[1, 1, 2, 0]$, $[0, 1, 1, 2]$ and $[1, 0, 1, 2]$.

These values are listed in Table I for the cumulative computation types with at least one successful score vector for one of the schemes. Particularly striking are the last three rows that correspond to cases with very few computations completed, i.e., when at most one worker completes all its assigned tasks. In these cases, CCPR is much more likely to recover $\mathbf{W}\boldsymbol{\theta}$; and hence, the computation deadline can be reduced significantly. For a more explicit comparison of the completion time statistics, we can analyze the probability of each type under a specific computation time statistics. Then, the probability of cumulative computation type $\mathbf{N} = (N_r, \dots, N_0)$ at time t is given by

$$\Pr(\mathbf{N}(t) = \mathbf{N}) = \prod_{s=0}^{r} P_s(t)^{N_s}, \tag{5}$$

where $P_s(t)$ is the probability of completing exactly s computations by time t. Let T denote the recovery time of the desired computation. Accordingly, for any of the schemes, we can write $\Pr(T < t) = \sum_{i=1}^{9} n_a(\mathbf{N}_i) \cdot \Pr(\mathbf{N}(t) = \mathbf{N}_i)$, $a \in \{m, u, c\}$, where the types $\mathbf{N}_i$ and corresponding $n_a(\mathbf{N}_i)$, $i = 1, \dots, 9$, are listed in Table I. It is now clear that CCPR has the highest $\Pr(T < t)$ for any t; and hence, the minimum average completion time $E[T]$. In the next subsection, we will highlight the partial recovery property of CCPR.

Cumulative computation type	MCC $n_m(\mathbf{N}_i)$	UC-MM $n_u(\mathbf{N}_i)$	CCPR $n_c(\mathbf{N}_i)$
$\mathbf{N}_1 : N_2 = 4, N_1 = 0, N_0 = 0$	1	1	1
$\mathbf{N}_2 : N_2 = 3, N_1 = 1, N_0 = 0$	4	4	4
$\mathbf{N}_3 : N_2 = 3, N_1 = 0, N_0 = 1$	4	4	4
$\mathbf{N}_4 : N_2 = 2, N_1 = 2, N_0 = 0$	6	6	6
$\mathbf{N}_5 : N_2 = 2, N_1 = 1, N_0 = 1$	12	12	12
$\mathbf{N}_6 : N_2 = 2, N_1 = 0, N_0 = 2$	6	6	6
$\mathbf{N}_7 : N_2 = 1, N_1 = 3, N_0 = 0$	0	4	4
$\mathbf{N}_8 : N_2 = 1, N_1 = 2, N_0 = 1$	0	12	12
$\mathbf{N}_9 : N_2 = 1, N_1 = 1, N_0 = 2$	0	8	8
$\mathbf{N}_{10} : N_2 = 0, N_1 = 4, N_0 = 0$	0	1	1
$\mathbf{N}_{11} : N_2 = 0, N_1 = 3, N_0 = 1$	0	4	4

TABLE II: Number of successful score vectors for each cumulative computation type that can result in the recovery of at least 3 out of 4 computations with $K = 4$ and $r = 2$.

C. Partial computation performance

Now, we compare the three schemes when we reduce the tolerance level, and aim at recovering only a portion of the computation results. In particular, for the above example, we will deem a scheme successful if it recovers at least 3 out of 4 values, $\{\mathbf{W}_1\boldsymbol{\theta}, \dots, \mathbf{W}_4\boldsymbol{\theta}\}$, corresponding to a tolerance of 25%. For each cumulative computation type the number of successful score vectors are listed in Table II. We can see that UC-MMC and CCPR have the same average completion time statistics. Hence, CCPR can provide a lower average per-iteration completion time for accurate computation compared to UC-MMC, while achieving the same performance when partial computation is allowed.

III. Design Principles of CCPR

For the encoding of the assigned computations we use a similar strategy to *rateless codes*, particularly to LT codes [47]. We first briefly explain the LT code structure, and highlight the required modification for our problem setup.

Consider a sequence of symbols $\overline{\mathbf{W}} = \{\mathbf{W}_1, \dots, \mathbf{W}_K\}$ (in our setup these correspond to submatrices $\mathbf{W}_i$) to be transmitted over an erasure channel which correspond to stragglers in

Algorithm 1 RCS coded computation

Data assignment phase:
$L = \sum_{i=1}^{m} d_i$
Choose random subset $\mathcal{I} \subset [K]$, $|\mathcal{I}| = L$
for row index $i = 1, \ldots, L$ **do**
    Randomly choose $j \in \mathcal{I}$
    Update $\mathcal{I}$: $\mathcal{I} \leftarrow \mathcal{I} \setminus \{j\}$
    $\mathbf{A}(i,:) = circshift(\overline{\mathbf{W}}, j-1)$
Data encoding phase:
for worker $k = 1, \ldots, K$ **do**
    for message $j = 1, \ldots, r$ **do**
        Starting row index: $l_s = \sum_{i=1}^{j-1} d_i + 1$
        Ending row index: $l_e = \sum_{i=1}^{j} d_i$
        $\widetilde{\mathbf{W}}_{i,j} = \sum_{l=l_s}^{l_e} \mathbf{A}(l,i)$

our model. The codewords (coded computations in our model) are formed as linear combinations of $\mathbf{W}_1, \ldots, \mathbf{W}_K$, and the goal is to correctly recover the original sequence from only a random subset of the coded symbols. In the encoding phase, a coded symbol is formed by choosing d elements randomly from $\overline{\mathbf{W}}$ and summing them, where d, which simply defines the degree of the symbol, comes from a distribution $P(d)$. In the decoding part, each coded symbol is decomposed by using the recovered symbols, that is if a coded symbol contains a previously recovered symbol then it is subtracted from the coded symbol to obtain a new coded symbol with a smaller degree. Overall the objective is to recover all K symbols from $K(1+\epsilon)$ coded symbols with as small an ϵ as possible which reflects the overhead.

We remark that coded symbols with smaller degrees can be decomposed faster; however, having many coded symbols with smaller degrees increases the probability of linear dependence among codewords. Hence, the degree distribution plays an important role in the performance of LT codes. It has been shown that for a carefully chosen $P(d)$, ϵ goes to zero as $K \to \infty$. LT codes have the following drawbacks when employed in distributed computation.

First, the LT codes are designed under the assumption of receiving a large number of coded symbols; however, in a distributed computation scenario the number of symbols is typically limited since each symbol corresponds to a message transmitted to the PS over the network, and increasing the number of messages may lead to congestion and communication delays at some point. On the other hand, for small K the overhead ϵ might be high.

Second, the degree distribution of LT codes, $P(d)$, is designed for the correct recovery of the original sequence $\overline{\mathbf{W}}$. However, in our partial coded computation scenario we want to provide a certain flexibility by allowing the recovery of only $(1-q)K$ symbols, for some predefined tolerance q. Although the partial recovery of the rateless codes has been studied in the literature [48], we note that partial recovery for coded computation requires a tailored approach since the computational tasks, each of which corresponding to a distinct coded symbol, are executed sequentially; thus, the corresponding erasure probabilities due to straggling behavior of workers, are neither identical nor independent. Therefore, the coded symbols must be designed taking into account their execution orders to prevent overlaps and to minimize the average completion time.

Hence, the main idea behind the CCPR scheme is to utilize the LT code structure in a more systematic way such that the degree of the each coded computation task is chosen carefully based on its computation order and with aim of partial recovery.

A. Computation order and degree limitation

We want to re-emphasize that, coded computation tasks with lower degrees can be recovered faster but as the number of computations received at the PS increases lower degree computations become less and less informative. Therefore, we want initial computations to be easily recoverable while those completed later to be more informative. Hence, we introduce the following design criteria; (i) for the first row of the computation assignment matrix, we consider uncoded computations, (ii) for a particular worker and computation orders $i < j < r$, the degree of the computation at order i can not be higher than that of the one at order j.

B. Uniformity imposed encoding

As highlighted before, coded messages with lower degrees may result in duplicate recoveries, wasting the computation resources. To this end, under the specified degree limitation, the main design challenge is how to form the coded computations to prevent duplicate messages as much as possible. Accordingly, the challenge is to distribute submatrices $\mathbf{W}_1, \ldots, \mathbf{W}_K$ among the coded computation tasks in a uniform fashion. We introduce two types of uniformity to help reduce the complexity of code design by introducing some structure to the code.

1) Order-wise uniformity: By order-wise uniformity, we impose a constraint on the code construction such that computations with the same order must have the same degree, and among the computations with the same order, each submatrix $\mathbf{W}_k$ must appear in exactly the same number of computations. Formally speaking, let

$$\widetilde{\mathcal{W}}_k^j \triangleq \left\{ \widetilde{\mathbf{W}}_{i,j} : \alpha_{j,k}^i \neq 0,\ i \in [K] \right\} \tag{6}$$

be the set of coded computations at order $j \in [r]$ containing submatrix $\mathbf{W}_k$. Then, the order-wise uniformity constraint imposes

$$|\widetilde{\mathcal{W}}_k^j| = d_j,\ \forall j \in [r],\ \forall k \in [K], \tag{7}$$

for some d_j.

2) Worker-wise uniformity: Worker-wise uniformity imposes a constraint on the coded computations assigned to each worker, such that the coded computations do not contain any common submatrices. Formally speaking, for any worker $i \in [K]$, if $\alpha_{j,k}^i \neq 0$, for some $j \in [r]$ and $k \in [K]$, then $\alpha_{l,k}^i = 0$, $\forall\ l \in [r] \setminus \{j\}$.

Next, we introduce an encoding structure which ensures both order-wise and worker-wise uniformity.

$$\mathbf{A}_{RCS} = \begin{bmatrix} \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \dots & \mathbf{W}_{20} \\ \mathbf{W}_4 & \mathbf{W}_5 & \mathbf{W}_6 & \dots & \mathbf{W}_3 \\ \mathbf{W}_{11} & \mathbf{W}_{12} & \mathbf{W}_{13} & \dots & \mathbf{W}_{10} \\ \mathbf{W}_{15} & \mathbf{W}_{16} & \mathbf{W}_{17} & \dots & \mathbf{W}_{14} \\ \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 & \dots & \mathbf{W}_5 \\ \mathbf{W}_{18} & \mathbf{W}_{19} & \mathbf{W}_{20} & \dots & \mathbf{W}_{17} \end{bmatrix}$$

Fig. 2: Data assignment matrix for $K = 20$ and $\mathcal{I} = \{1, 4, 11, 15, 6, 18\}$.

$$\begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_4 \\ \mathbf{W}_{11} \\ \mathbf{W}_{15} \\ \mathbf{W}_6 \\ \mathbf{W}_{18} \end{bmatrix} \rightarrow \mathbf{C}_1 = \begin{bmatrix} \widetilde{\mathbf{W}}_{1,1} \\ \widetilde{\mathbf{W}}_{1,2} \\ \widetilde{\mathbf{W}}_{1,3} \end{bmatrix} = \begin{bmatrix} \mathbf{W}_1 \\ \mathbf{W}_4 + \mathbf{W}_{11} \\ \mathbf{W}_{15} + \mathbf{W}_6 + \mathbf{W}_{18} \end{bmatrix}$$

Fig. 3: Illustration of the encoding phase for the first worker and the corresponding first column of the computation assignment matrix, $\mathbf{C}_1$.

IV. Randomly Circular Shifted (RCS) Code Design

In this section, we introduce the randomly circular shifted (RCS) code design for coded distributed computation, which consists of two steps; namely data assignment and code construction. Before explaining these steps in detail, we first define the degree vector $\mathbf{d}$ of length r, whose ith entry d_i denotes the degree of the coded computations assigned to workers in order $i \in [r]$. Based on the aforementioned design criteria we set $d_1 = 1$ and, for any $i < j$, we have $d_i \leq d_j$. Once the degree vector $\mathbf{d}$ is fixed, the two phases of RCS code design can be implemented.

In the first phase, a data assignment matrix is formed by using random circular shifts on the vector of submatrices $\overline{\mathbf{W}} = [\mathbf{W}_1, \dots, \mathbf{W}_K]$. At the beginning of the first phase, an index set $\mathcal{I} \subset [K]$ of size $L = \sum_{i=1}^{r} d_i$ is randomly chosen, where L here denotes the total number of partial computations that can be recovered from a worker. Then using the elements of $\mathcal{I}$ as a parameter of the circular shift operator on vector $\overline{\mathbf{W}}$, a data assignment matrix $\mathbf{A}_{RCS}$ is formed following Algorithm 1 (line 5-8).

Let us illustrate the RCS code on a simple example. Consider $K = 20$ workers and $\mathbf{d} = [123]$. This means $L = 6$, and assume that $\mathcal{I} = \{1, 4, 11, 15, 6, 18\}$. Then, for the ith row of $\mathbf{A}_{RCS}$, $j \in \mathcal{I}$, is chosen randomly and discarded from $\mathcal{I}$, while $\overline{\mathbf{W}}$ is circularly shifted by $j - 1$. For the sake of simplicity, we assume that elements of $\mathcal{I}$ are chosen with the given order $1, 4, 11, 15, 6, 18$, and the corresponding data assignment matrix is illustrated in Fig. 2.

Once the data assignment matrix is fixed, codewords can be generated based on the degree vector $\mathbf{d}$ for each column independently and identically. The colors in the data assignment matrix in Fig. 2 represent the submatrices that will form the same coded computation. The computation assignment for the first user, using the submatrices on the first column of the data assignment matrix, is illustrated in Fig. 3.

We note that each coded message corresponds to a linear equation, and in the decoding phase any approach for solving a set of linear equations can be utilized, e.g., we can form a matrix with the coefficients of the coded messages, $\alpha^i_{j,k}$, that is, each coded message is represented by a binary row vector, and obtain the *reduced row echelon* form. Similarly to LT codes, we consider a low complexity decoding framework that decomposes the codewords successively by using only recovered symbols. From the construction, a codeword of degree d is reduced $d - 1$ time to recover an input symbol. Also, since the first K codewords are of degree one, the total number of reduction on the codewords is limited by $K \times \tilde{L}$ where $\tilde{L} = \sum_{i=2}^{r} d_i - 1$; and hence, the worst case complexity of the decoding phase is $O(K\tilde{L})$.

V. Extension to coded communication scenario

Above, we have mainly focused on coded computation in the context of a linear regression problem, where the main computation task boils down to distributed matrix-vector multiplication. In a more general distributed computation problem, in which the computations cannot be expressed as a linear transform of the dataset, we cannot employ a similar coded computation technique. However, if the overall computation task can be written as the summation of smaller partial computation tasks, then, the redundancy can be achieved by assigning each of these partial computations to multiple workers. Communication load of such an implementation can be reduced by coded communication, where each worker sends to PS linear combinations of its partial computations. The gradient coding (GC) scheme, introduced in [8], considers gradient estimates on subsets of a dataset as partial computations, and achieves redundancy by replicating parts of the dataset at multiple workers. This approach has been extended in various directions to improve the performance [9]–[14].

Let $\mathcal{G} = \{\mathbf{g}_1, \dots, \mathbf{g}_K\}$ be the results corresponding to K partial computations, and the goal is to recover their sum at PS. In the GC scheme with computation load r, r partial computations, denoted by $\mathcal{G}_k$, are assigned to worker k. Each worker, after completing r partial computations, sends a linear combination of its results to the PS

$$\mathbf{c}_k \triangleq \mathcal{L}_k(\mathbf{g}_i : \mathbf{g}_i \in \mathcal{G}_k). \tag{8}$$

We refer to the linear combinations $\mathbf{c}_1, \dots, \mathbf{c}_K$ as *coded partial computations*. The PS waits until it receives sufficiently many coded partial computations to recover the full gradient. It is shown in [8] that, for any set of non-straggler workers $\tilde{\mathcal{K}} \subseteq [K]$ with $|\tilde{\mathcal{K}}| = K - r + 1$, there exists a set of coefficients $\mathcal{A}_{\tilde{\mathcal{K}}} = \{a_k : k \in \tilde{\mathcal{K}}\}$ such that

$$\sum_{k \in \tilde{\mathcal{K}}} a_k c_k^{(t)} = \sum_{k=1}^{K} g_k^{(t)}. \tag{9}$$

The GC scheme is designed for exact recovery of the summation, and limits the number of messages per worker to one. The MMC variation of GC is studied before in [11], [13]. Here, we will show that the RCS code proposed for matrix-vector multiplication can also be used for partial recovery in

$$\mathbf{A}_{RCS} = \begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \cdots & \mathbf{g}_{20} \\ \mathbf{g}_4 & \mathbf{g}_5 & \mathbf{g}_6 & \cdots & \mathbf{g}_3 \\ \mathbf{g}_{11} & \mathbf{g}_{12} & \mathbf{g}_{13} & \cdots & \mathbf{g}_{10} \\ \mathbf{g}_{15} & \mathbf{g}_{16} & \mathbf{g}_{17} & \cdots & \mathbf{g}_{14} \\ \mathbf{g}_6 & \mathbf{g}_7 & \mathbf{g}_8 & \cdots & \mathbf{g}_5 \\ \mathbf{g}_{18} & \mathbf{g}_{19} & \mathbf{g}_{20} & \cdots & \mathbf{g}_{17} \end{bmatrix}.$$

Fig. 4: Data assignment matrix for $K = 20$ and $\mathcal{I} = \{1, 4, 11, 15, 6, 18\}$.

$$\begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_4 \\ \mathbf{g}_{11} \\ \mathbf{g}_{15} \\ \mathbf{g}_6 \\ \mathbf{g}_{18} \end{bmatrix} \rightarrow \begin{bmatrix} \widetilde{\mathbf{g}}_{1,1} \\ \widetilde{\mathbf{g}}_{1,2} \\ \widetilde{\mathbf{g}}_{1,3} \end{bmatrix} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_4 + \mathbf{g}_{11} \\ \mathbf{g}_{15} + \mathbf{g}_6 + \mathbf{g}_{18} \end{bmatrix}.$$

Fig. 5: Illustration of the encoding phase at the first worker for coded communication with the computation assignment matrix in Fig. 4 and a degree vector $\mathbf{d} = [1, 2, 3]$.

coded communication with a small variation in the encoding phase.

We will illustrate RCS coded communication on an example. Consider the previous example with $K = 20$ workers. As before, an $L \times K$ computation assignment is formed to assign partial computations to workers with a certain computation order as illustrated in Fig. 4. In the coded communication scenario encoding takes place after the computation, therefore the computation load of the computation assignment matrix in Fig. 4 is $r = L = 6$, whereas the same matrix would have a computation load of $r = 3$ in the coded computation scenario. Again, once the assignment matrix is formed, coded messages for each worker are constructed according to the given degree vector $\mathbf{d}$ and based on the assignment matrix $\mathbf{A}_{RCS}$ as illustrated in Fig. 5. Note that, similarly to the coded communication scenario RCS code allows recovery of only a subset of the partial computations, and can compute an approximation to the required summation. Note, however, that, while in coded computation missing results correspond to entries of the vector we would like to compute, here the missing results will impact every entry of the desired computation as we will be missing some of the the partial computations. We want to remark that, concurrently to our work, GC scheme with particular focus on the trade-off between computation accuracy and time has been studied in [14], [40]–[42], [49].

VI. Generalized RCS codes

In the introduced RCS code structure, the main computation task is divided into K equal sub-tasks. However, if the variation on the computational speed of the workers is small, it might be better to divide them into even smaller tasks in order to better utilize the computational resources

Here, we present generalized RCS codes that allow adjusting the sizes of individual computation tasks. In generalized RCS codes, the encoding part remains the same as before, but the construction of the $\mathbf{A}_{RCS}$ matrix is as follows. First, $\mathbf{W}$ is divided into KN disjoint submatrices, i.e., $\mathbf{W}_1, \ldots, \mathbf{W}_{KN}$, which are then divided into N groups $\overline{\mathbf{W}}^{(1)}, \ldots, \overline{\mathbf{W}}^{(N)}$, each containing K submatrices, i.e., $\overline{\mathbf{W}}^{(i)} = [\mathbf{W}_{(i-1)K+1}, \ldots, \mathbf{W}_{iK}]$. Before the construction of $\mathbf{A}_{RCS}$ matrix, we will define a vector $\mathbf{z}$ which will be useful.

Let $\mathbf{z}$ is a $L = \sum_{i=1}^{m} \mathbf{m}(i)$ dimensional vector where each entry is from the set $[N]$. Construction of $\mathbf{A}_{RCS}$ is executed row by row such that for the ith row, we first check $z(i)$ and accordingly use the submatricies in group $\overline{\mathbf{W}}^{(Z(i))}$. Once we decide on the group of submatricies $\overline{\mathbf{W}}^{(Z(i))}$, we randomly sample an element j from set $\in \mathcal{I}_{z(i)}$ and circularly shift the $\overline{\mathbf{W}}^{(Z(i))}$ with $j-1$ before assigning it as the ith row of $\mathbf{A}_{RCS}$. We remark that, initially, $\mathcal{I}_i = [K]$, $\forall i \in [N]$, and after the circularshift operation based on sampled j we remove the j from the corresponding set $\mathcal{I}_i$ to prevent repetition among the rows. The detailed procedure is illustrated in Algorithm 2. We present a simple example for $K = 4$ and $N = 2$ to clarify the overall procedure. Let $\mathcal{I}_1 = \{1, 3\}$, $\mathcal{I}_2 = \{1, 4, 3\}$, and $\mathbf{z} = [2, 1, 1, 2, 2]$, and the construction procedure of $\mathbf{A}_{RCS}$ is illustrated in Fig. 6. The computation assignment matrix $\mathbf{C}_{RCS}$ is given for $\mathbf{d} = [1, 1, 3]$ in Fig. 7. As in the previous example illustrated in Fig. 3, each user is allowed to send at most 3 messages, but now the computation load is $r = 3/2$ instead of 3 as each of the computations is half the size of those in the previous example. If the computation speeds of the workers are more likely to be similar, it will be better to divide the main computation task into smaller subtasks to utilize the available computation resources better. Here, we remark that the first row of $\mathbf{C}_{RCS}$ is from $\overline{\mathbf{W}}^{(2)}$ while the second row is from $\overline{\mathbf{W}}^{(1)}$. Hence, considering only the first two assigned computations, the recovery probability of $\mathbf{W}_i\boldsymbol{\theta}$ is higher for $\mathbf{W}_i \in \overline{\mathbf{W}}^{(2)}$, compared to $\mathbf{W}_i \in \overline{\mathbf{W}}^{(1)}$. Therefore, the third row is generated using two submatrices from $\overline{\mathbf{W}}^{(2)}$ and one submatrix from $\overline{\mathbf{W}}^{(1)}$. Consequently, by playing with $\mathbf{d} = [1, 1, 3]$ and $\mathbf{z} = [2, 1, 1, 2, 2]$ different operating points in terms of the computation speed and accuracy can be achieved.

Algorithm 2 Generalized RCS coded computation

Data assignment phase:
$L = \sum_{i=1}^{m} d_i$
for $j = 1 : N$ **do**
  $\mathcal{I}_j = [K]$
for row index $i = 1, 2, \ldots, L$ **do**
  Randomly choose $j \in \mathcal{I}_{z(i)}$
  Update $\mathcal{I}_{z(i)}$: $\mathcal{I}_{z(i)} \leftarrow \mathcal{I}_{z(i)} \setminus \{j\}$
  $\mathbf{A}(i, :) = circshift(\overline{\mathbf{W}}^{(z(i))}, j - 1)$
Data encoding phase:
for worker $k = 1, 2, \ldots, K$ **do**
  for message $j = 1, \ldots, r$ **do**
    Starting row index: $l_s = \sum_{i=1}^{j-1} d_i + 1$
    Ending row index: $l_e = \sum_{i=1}^{j} d_i$
    $\widetilde{\mathbf{W}}_{k,j} = \sum_{l=l_s}^{l_e} \mathbf{A}(l, k)$

$$\mathbf{A}_{RCS} = \begin{bmatrix} \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 \\ \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 & \mathbf{W}_4 & \mathbf{W}_1 & \mathbf{W}_2 \\ \mathbf{W}_8 & \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 \\ \mathbf{W}_7 & \mathbf{W}_8 & \mathbf{W}_5 & \mathbf{W}_6 \end{bmatrix}$$

Fig. 6: $\mathbf{A}_{RCS}$ based on $\mathbf{z} = [2, 1, 1, 2, 2]$, and random samples $\{1, 3\} \in \mathcal{I}_1$, $\{1, 4, 3\} \in \mathcal{I}_2$

.

$$\mathbf{C}_{RCS} = \begin{bmatrix} \mathbf{W}_5 & \mathbf{W}_6 & \mathbf{W}_7 & \mathbf{W}_8 \\ \mathbf{W}_1 & \mathbf{W}_2 & \mathbf{W}_3 & \mathbf{W}_4 \\ \mathbf{W}_3 + \mathbf{W}_8 + \mathbf{W}_7 & \mathbf{W}_4 + \mathbf{W}_5 + \mathbf{W}_8 & \mathbf{W}_1 + \mathbf{W}_6 + \mathbf{W}_5 & \mathbf{W}_2 + \mathbf{W}_7 + \mathbf{W}_6 \end{bmatrix}.$$

Fig. 7: $\mathbf{C}_{RCS}$ based on $\mathbf{A}_{RCS}$ and $\mathbf{d} = [1, 1, 3]$

VII. Connection to LT Codes and Choosing the Degree Distribution

In this section, we will discuss the connections of the proposed RCS coding strategy to LT codes to better understand how the degree distribution can be optimized. First, we briefly summarize the way decoding process is analyzed in the LT code design [47], [50]. In LT codes, the data we want to recover is divided in to K parts, each of them referred to as an *input symbol*. The *output symbols* are generated as XORed combinations of the d randomly chosen input symbols, where d is a random variable from a discrete distribution $\Omega(d)$. Input symbols XORed to obtain an output symbol are called its *neighbors*. LT process consists of three key steps;

- (release) When an output symbol of degree one appears, we refer to it as a release.
- (cover) Following release of an output symbol, the corresponding unique input symbol is covered.
- (process) A covered input symbol is removed from all its neighboring output symbols.

All the covered but unprocessed symbols are stored in a buffer called *ripple*. During the flow of the LT process, initially all degree-one codewords are immediately released and stored in the ripple, then at each step of the process, one symbol is chosen to be processed until the ripple size, denoted by R, becomes zero, at which point the algorithm halts.

The evolution of the ripple size throughout the decoding iterations is the main focus of the LT code design when defining the degree distribution, and the aim is to keep the ripple size as constant as possible to minimize the overhead. Ideal strategy is to have $R = 1$ thoughout the LT process [47], however, in practice, due to randomness ripple can be emptied before the K input symbols are covered. Hence, in [47], *robust soliton distribution* is introduced for choosing the degree of the output symbols so that LT process starts with ripple size $R > 1$ and kept constant as much as possible.

In the next two subsections, we first discuss the impact of the straggling behavior; that is, the statistics of the codewords with particular order, on the design of the degree distribution in our scenario.

A. The impact of the straggling behaviour

In the scope of this work, to model the statistics of the computational latency we will employ the shifted exponential distribution, similar to the existing works such as [15], which is defined with two parameters, the mean of the exponential distribution $1/\mu$ and the shift parameter α such that the probability of completing s computations at any server, e.g., performing s identical matrix vector multiplications, by time t is given by

$$F_s(t) \triangleq \begin{cases} 1 - e^{-\mu(\frac{t}{s} - \alpha)}, & \text{if } t \geq s\alpha, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

We emphasize that since the α parameter simply denotes the minimum required time to complete a single computation and as we increase it, while keeping μ constant, computation latency of the workers become more and more correlated. Consequently, if we use $0 \leq \rho_i \leq 1$ to denote the fraction of the codewords with order i collected at the PS, then ρ_1 increases in parallel to increase in the shift parameter. We remark that, although this analysis is not limited to shifted exponential distribution we particularly use it here, since it clearly reflects the impact of the straggling behaviour on the code design. Now, we consider a simulation setup where we assume $K = 40$ and $r = 3$ and use the shifted exponential model for the computational statistics with $\mu = 10$ and 4 different α values, $0.01, 0.05, 0.1$ and 0.2, respectively. In Table III, we illustrate the fraction the codewords with a particular order within the first K received messages. At this point we want to highlight that the major alteration compared to LT code is to utilize the side information regarding the erasure probability in the design of the degree distribution. To be more precise, in robust solution distribution, probability of degree one codewords kept small in order to prevent reception of duplicate symbol, however RCS further utilizes the degree one codewords since degree one codewords from different worker do not overlap. To clarify let $\tilde{\Omega}_{(K)}(d)$ denote degree distribution observed at PS when K messages are received in total. In the case of RCS and shifted exponential distribution with parameters $\mu = 10$ and $\alpha = 0.01$ we observe $\tilde{\Omega}_{(K)}(1) = 0.55$. At this point we want to highlight the following fact that with RCS there is no overlapping degree

Shift parameter	Order of the Codewords		
	ρ_1	ρ_2	ρ_3
$\alpha = 0.01$	0.55	0.3	0.15
$\alpha = 0.05$	0.69	0.2750	0.0375
$\alpha = 0.1$	0.78	0.22	0
$\alpha = 0.2$	0.9	0.1	0

TABLE III: Fraction of the codewords with a particular order with respect to different shift parameters, α.

one codewords, however, using the same distribution for the LT codes induce on average $K/10$ number of overlapping degree one codewords which implies around %10 redundancy. The use of degree one codewords is one of the major difference between the LT code design and the RCS.

B. Incorporating robust soliton distribution

To provide a general perspective on the impact of the degree distribution, we first focus on one-shot recovery such that the recovery of a symbol from a received codeword of degree d, given that ρ portion of the symbols are already recovered. Based on the results illustrated in Table III, we are particularly interested in a regime where $\rho > 0.5$.

Given values of K, ρ and d, probability of recovering an input symbol from a codeword/output symbol of degree d is

$$\frac{\binom{K\rho}{d-1}\binom{K-K\rho}{1}}{\binom{K}{d}} = \Pi_{i=1}^{d-2}\frac{K\rho - i}{K-i}\frac{d(K-K\rho)}{K-d-1}, \tag{11}$$

further if K is sufficiently large, also assuming $K\rho >> d$, this can be approximated as

$$\rho^{d-1}d(1-\rho). \tag{12}$$

Hence, for a given ρ, one can choose d that maximizes the (11) as a greedy approach. This can be considered as the optimal degree choice under the assumption that codeword is discarded immediately if an input symbol can not be recovered. However, in LT process codewords are buffered and might be used to recover an input symbol at any step of the decoding phase. Hence, we consider d that maximizes maximizes the (11) as a lower bound and denoted with d_{min}. One can easily observe that d_{min} increases with respect to ρ, e.g., for R values around 0.5, 0.6, 0.7 and 0.8, d_{min} takes values around $2, 3, 4$ and 5, respectively. Hence, we can emphasize that there is no *once-for-all* solution and the degree distribution should be designed based on the computation statistics. The reason we define d_{min} as the lower bound for the codeword degree is that, although it maximize the probability of recovering a new input symbol in a one-shot manner, it also give rise to redundancy that is the probability of a codeword having all its neighbours from the R recovered input symbols. This probability can be approximated as ρ^d.

To understand the dynamic nature of the decoding process we revisit the analysis of the robust soliton distribution [47]. The objective of the robust soliton distribution is to initially obtain a small but sufficient ripple size R from degree one codewords and choose the degree distribution for $\Omega(d)$ for $d > 2$ in a way to keep ripple size constant through the decoding process. Hence, to keep the ripple size constant the number of released symbol should be kept around one, on average. Given that the ripple size is R and the number of unprocessed symbols is L, the probability of a codeword of degree d being released and an input symbol being added to ripple is equal to

$$q(d, L, R) = \frac{\binom{K-(L+1)}{d-2}\binom{L-R}{1}}{\binom{K}{d}} \tag{13}$$

$$= \frac{d(d-1)(L-R)\Pi_{i=0}^{d-3}(K-(L+1)-i)}{\Pi_{i=0}^{d-1}(K-i)}. \tag{14}$$

Hence, robust soliton distribution aims to keep density of the codewords with degree d proportional to $\frac{1}{d(d-1)(L-R)}$ so that at each step, on average, one symbol is released and added to ripple. As we already aforementioned, the key difference between the RCS and the LT code design is the density of the degree one codewords, which is also the reason why it is not possible to follow the same logic used in the robust soliton distribution in this setup. To be more precise, the RCS strategy implies a large initial ripple size R compared to LT codes. Here, we remark that when a symbol is released, probability of released symbol being added to ripple is $L - R/L$, hence if one try to keep R constant, then large portion of the released symbols will be redundant.

Hence, we slightly change the analysis of the robust soliton distribution in the following way; first, as in the original analysis, all degree one codewords are added to the ripple immediately, then instead of one-by-one processing, we process all $R = \rho K$ symbols at the same time, thus R is reduced to 0. Then, new released symbols, following the process of $R = \rho K$ symbols, are added to ripple. At this point we use the equation (11) to compute estimate of the new ripple size. Then, we follow analysis of robust soliton distribution by processing exactly one symbol at each time and try to keep R as much as constant. By achieving new R value that is comparatively smaller than ρK the redundancy is avoided.

In case of robust soliton distribution the ripple size R is only depend on the density of degree one output symbols, but now in our case R is actually depends on the whole degree distribution $\Omega(d)$ such that R will be bounded by

$$R \leq K(1-\rho)\sum_{d=2}^{d_{max}}\Omega(d)\rho^{d-1}d(1-\rho) \tag{15}$$

$$\rho_{rip} \leq \underbrace{(1-\rho)^2\sum_{d=2}^{d_{max}}\Omega(d)\rho^{d-1}d}_{\tilde{\rho}_{rip}},$$

where ρ_{rip} is the ratio of the symbols at the ripple. Although (15) gives an upper bound on ρ_{rip}, this upper bound is tight, $\rho_{rip} \approx \tilde{\rho}_{rip}$, when $\tilde{\rho}_{rip} << 1-\rho$ and K sufficiently large. This is equivalent to observing that when R balls are chosen out of L with replacement, the number of distinct balls are approximately equal to R for R/L is sufficiently small. Therefore, we ignore the redundancy within the released symbols, but focus on the redundant output symbols whose all

neighbours are among the initially released ρK input symbols. The ratio of such output symbols can be approximated as

$$\rho_{red} = (1-\rho) \sum_{d=2}^{d_{max}} \Omega(d)\rho^d. \tag{16}$$

In the case of full recovery this implies that at least $K\rho_{red}$ extra codewords are required. To exemplify numerically, when $\rho = 0.55$, assuming all the remaining codewords are of degree two gives $\rho_{red} = 0.135$, and assuming degree three gives $\rho_{red} = 0.0745$. Hence, although robust soliton distribution has a peak around $d = 2$, in our case we expect $\Omega(d)$ has a peak at a higher d values to avoid redundancy. In overall, there are three critical aspects when deciding the degree distribution, particularly to target the full recovery, keeping ρ_{red} as close to zero as possible, having small but sufficient ripple ratio ρ_{rip} and keeping $\sum_{d=2}^{d_{max}} \Omega(d)q(d,L,R)$ close to $\frac{1}{K\rho}$ in order to preserve the ripple size. In Fig. 8, we illustrate the probability of symbol being released from a codeword of degree d and added to the ripple for values ρ_{rip} of $0.05, 0.075$, and 0.1, respectively.

Hence, if we revisit our particular scenario where $r = 3$ and $\alpha = 0.01$ and target the full recovery then, based on the aforementioned key aspects, a proper choice for degree distribution would be $\mathbf{d} = [1\ 4\ 8]$ or $\mathbf{d} = [1\ 5\ 7]$, also taking into account that $\rho_2 = 0.3$ and $\rho_3 = 0.15$. Once, the best possible degree distribution for the full recovery is available, then based on the requirements, one can search for the appropriate degree distribution by reducing degree values while still trying to keep ρ_{rip} as small as possible. In Table IV, we illustrate this trade-off by comparing the total number of messages received until the termination with a tolerance q.

C. Changing the decoding strategy for a better trade-off

In the previous subsection, we highlighted the main aspects on the design of degree distribution when the LT process is utilized for decoding. However, it might be possible to achieve better trade-offs by altering the decoding strategy as partly discussed in [48]. Gaussian elimination method can be used instead of LT process, but in general it is a computationally costly procedure. Nevertheless, a hybrid approach can be utilized to achieve better decoding performance without excessive computational overhead. That is, at the beginning LT process is followed to recover input symbols and when it halts Gaussian elimination is applied to remaining codewords. The key advantage of such strategy will be reducing the redundancy. Next, we illustrate this trade-off with an example in our setup. As demonstrated in Table V, change of the decoding process offers better trades-offs, especially when higher degree codewords, e.g., $\mathbf{d} = [1\ 5\ 7]$, are used when it is desired to reduce the number of received messages at the operating point $q = 0$, but still want to achieve lower latency at $q = 0.3$ and $q = 0.1$. In the case of LT process, to achieve similar performance at the regime of $q = 0.3$ and $q = 0.15$, it is required to reduce degree of codewords, such as using $\mathbf{d} = [1\ 3\ 5]$, which induce a visible overhead for the full recovery in terms of the number of received messages.

VIII. Numerical Results and Discussions

For the numerical analysis, we will first analyze the convergence performance of the partial recovery strategy for coded computation with RCS codes under different tolerance requirements, then we will compare the *average per-iteration completion time* of the RCS code with UC-MMC and MDS coded computation (MCC) schemes, and, finally we will extend our analysis to coded communication.

A. Simulation setup

For the simulations, we consider a linear regression problem over synthetically created training dataset, according to normal mixture distribution as in [18], consisting of size of 2000 samples. In the generation of synthetic data, we first create the true parameter model $\theta^\star$ randomly sampling each entry from the interval $[0, 1]$ according to uniform distribution. Once we have the $\theta^\star$, we construct two mean vectors $\mu_1 = \frac{1.5}{d}\theta^\star$ and $\mu_2 = \frac{-1.5}{d}\theta^\star$, where this mean vectors are later used in the normal mixture distribution $\frac{1}{2}\mathcal{N}(\mu_1, \mathbf{I}) + \frac{1}{2}\mathcal{N}(\mu_2, \mathbf{I})$ to generate the each row of dataset $\mathbf{X}$.

We assume that there are $K = 40$ homogeneous workers, and set $\mu = 10$ and $\alpha = 0.01$ for the statistics of their computation speeds in (10). For RCS coded computation we choose the degree vector $\mathbf{d} = [1, 2, 3]$, which corresponds to computation load of $r = 3$, and we execute Algorithm 1 accordingly. In all the simulations, we set the learning rate to $\lambda = 0.1$.

B. Simulation results

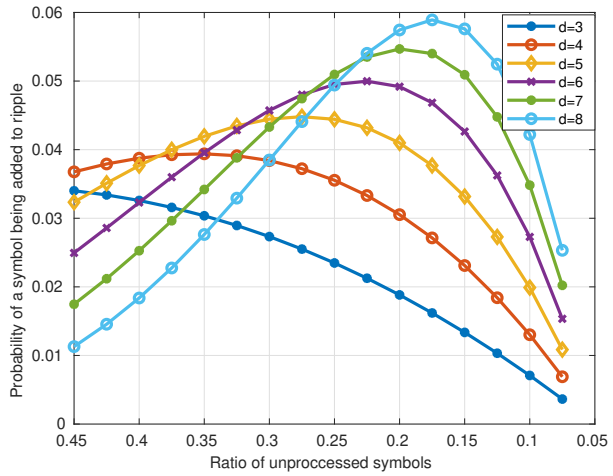
We first consider a model size of $d = 800$, and evaluate the training error over $T = 50$ iterations[1] for tolerance level of $q = 0$ (which corresponds to full recovery), $q = 0.15$, and $q = 0.3$. One can observe in Fig. 9 that, although the convergence speed reduces with increasing tolerance level at each iteration, partial recovery does not harm the convergence behaviour much, especially if the tolerance level is moderate, e.g., $q = 0.15$. We then repeat the same experiments for a model size of $d = 400$, which demonstrates similar trends as seen in Fig. 10.

What we want to see next is how much reduction in per-iteration time can be achieved by the partial recovery scheme.

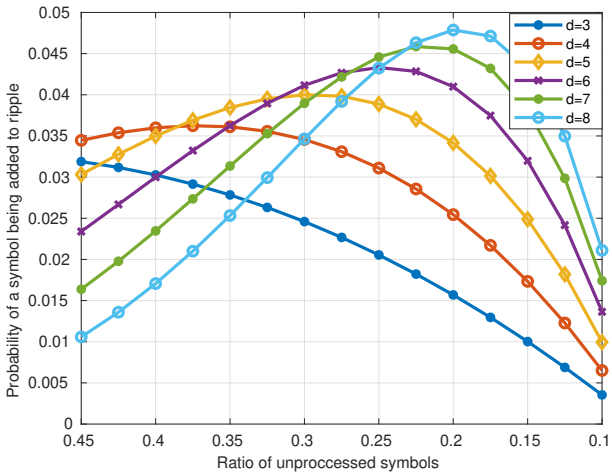
Hence, we present the per-iteration time of three different schemes, namely RCS, UC-MMC and MCC, for computational load $r = 3$. For RCS, we use the order vector $\mathbf{m} = [1, 2, 4]$, for UC-MMC we use cyclic shifted assignment as in [19], and finally, for MCC we use $(\lceil K/r \rceil, K) = (14, 40)$ MDS code.

We compare the three schemes in terms of two performance metrics: *average completion time* and *number of received messages*, which demonstrate how fast an iteration will be completed, and the induced communication load, respectively. From Table VI, one can observe that for full recovery, i.e., $q = 0$, RCS outperforms both UC-MMC and MCC schemes. We also observe that by allowing partial recovery it is possible

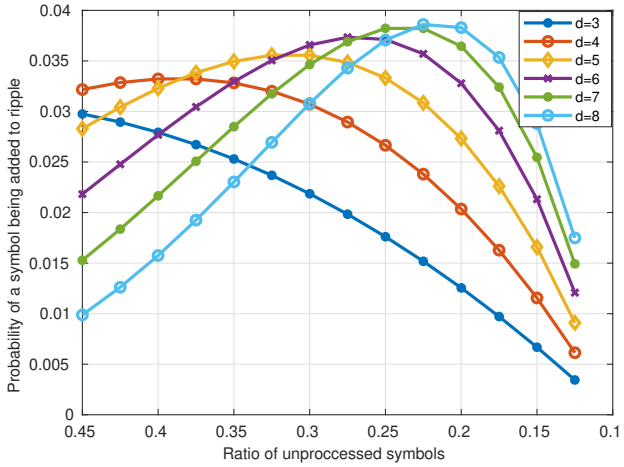
[1]For the convergence plot, we take average over 100 independent simulations.



(a) $\rho_{rip} = 0.05$.



(b) $\rho_{rip} = 0.075$



(c) $\rho_{rip} = 0.1$.

Fig. 8: The impact of the codeword degree on the ripple evaluation with respect to the initial ripple ratio ρ_{rip} for $\rho = 0.55$.

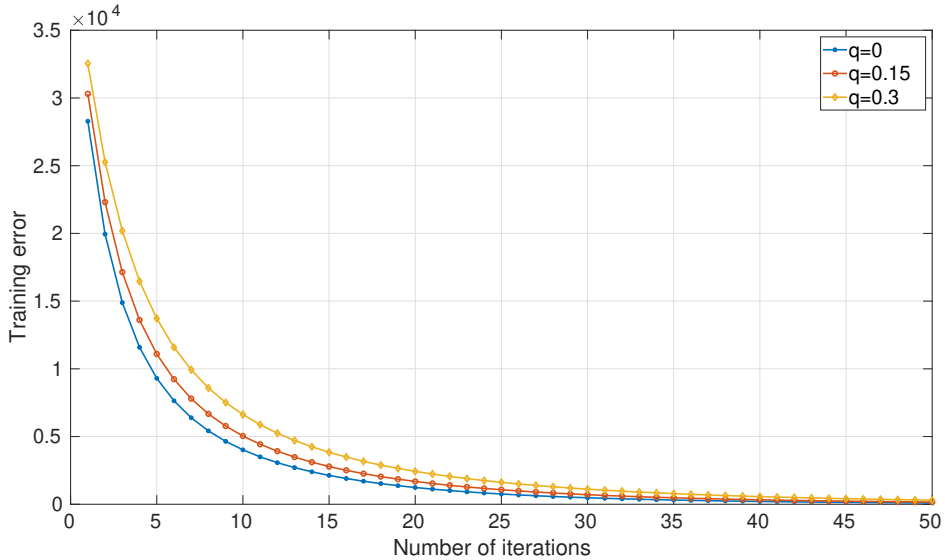


Fig. 9: Training error over $T = 50$ iterations, for a model size of $d = 800$, and tolerance level of $q = 0, 0.15, 0.3$, respectively.

Degree vector	Average Number of Codewords		
	$q = 0.3$	$q = 0.15$	$q = 0.0$
$\mathbf{d}$ = [1 2 3]	34	42.7	64.3
$\mathbf{d}$ = [1 3 3]	34.9	41.2	58.6
$\mathbf{d}$ = [1 3 4]	36	41.6	55.6
$\mathbf{d}$ = [1 3 5]	37	41.6	53.6
$\mathbf{d}$ = [1 4 4]	37.06	41.17	52.1
$\mathbf{d}$ = [1 5 7]	41.7	43.8	46.6
$\mathbf{d}$ = [1 4 8]	40.9	44.3	47.24

TABLE IV: Average number of required codewords to complete an iteration with a tolerance of q

Degree vector and decoding process	Average Number of Codewords		
	$q = 0.3$	$q = 0.15$	$q = 0.0$
$\mathbf{d}$ = [1 5 7] Hybrid	38.4	39.9	44.4
$\mathbf{d}$ = [1 5 7] LT process	41.7	43.8	46.65
$\mathbf{d}$ = [1 3 5] Hybrid	35.8	40.85	53
$\mathbf{d}$ = [1 3 5] LT process	37	41.6	53.6

TABLE V: Average number of required codewords to complete an iteration with a tolerance of q

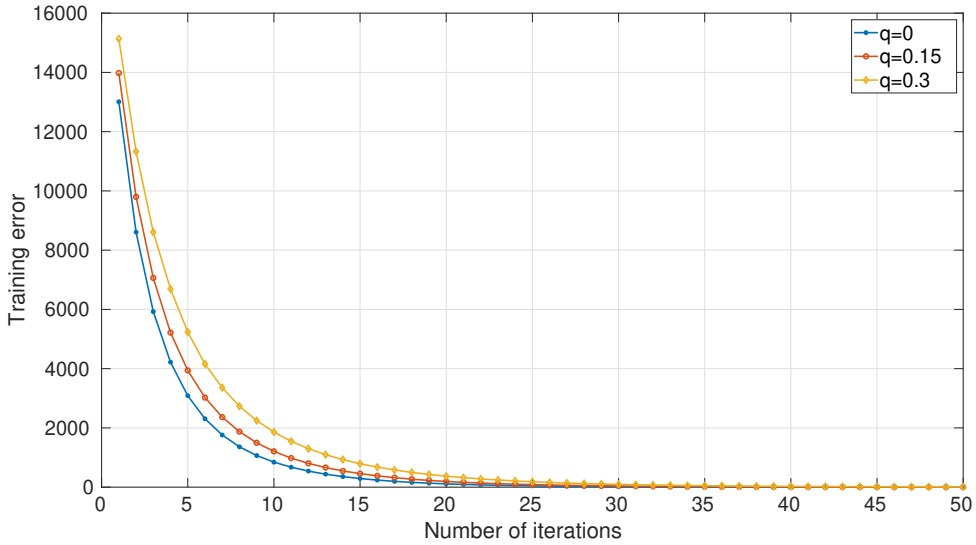


Fig. 10: Training error over $T = 50$ iterations, for a model size of $d = 400$, and tolerance level of $q = 0, 0.15, 0.3$, respectively

Computation strategy / tolerance	RCS			UC-MMC			MCC
	$q = 0.0$	$q = 0.15$	$q = 0.3$	$q = 0$	$q = 0.15$	$q = 0.3$	
Average iteration time	0.1475	0.0936	0.0776	0.2424	0.1170	0.0799	0.1572
Number of received messages	60.93	42.38	35.03	81.29	51.16	36.70	14

TABLE VI: Comparison of the proposed RCS scheme with UC-MMC and MCC schemes for the computational load $r = 3$

to achieve approximately 40% to 50% reduction in the per iteration completion time with $q = 0.15$ and $q = 0.30$, respectively. Here, we note that the partial recovery approach can be also employed with UC-MMC; however, as demonstrated in Table VI, RCS outperforms UC-MMC for all given tolerance values. Besides, with RCS PS completes an iteration with less number of received messages, which means that compared to UC-MMC, RCS induces less communication load and congestion. For instance, for $q = 0$ UC-MMC requires, on average, 28% more messages to complete an iteration compared to the proposed RCS scheme, which is shown to be a critical factor affecting the performance of real implementations [43]. We note that MCC requires the minimum number of messages to complete an iteration. Hence, for $q = 0$ MCC can be a better alternative; nevertheless, another advantage of RCS compared to MCC is that the decoding process can be executed in parallel to computations so that the additional latency due to decoding is minimized. Finally, based on the simulation results, we can conclude that the RCS scheme operates most efficiently when partial recovery is aimed with a low tolerance level, e.g., $q = 0.15$, since in this regime we can see the advantage of both using coded computation and partial recovery. To clarify this point, we observe that increasing the tolerance level to $q = 0.3$ makes considerable impact on the training accuracy, but the reduction on the average per-iteration computation time is relatively small. Besides, as q increases the performance of the UC-MMC scheme gets closer to that of RCS.

Computation strategy / tolerance rate	RCS			UC-MMC			GC
	$q = 0.0$	$q = 0.15$	$q = 0.3$	$q = 0$	$q = 0.15$	$q = 0.3$	
Average iteration time	0.2219	0.1231	0.0940	0.1874	0.0986	0.0736	1.2575
Number of received messages	62.56	41.55	32.37	99.63	55.06	38.30	35

TABLE VII: Comparison of the proposed RCS scheme with UC-MMC and GC schemes for the computational load $r = 6$

We also conduct an experiment for the generalized RCS code with $N = 2$. For the simulation we choose $d = [1, 1, 4, 8]$ so that 4 coded computations are assigned to each worker in total and the complexity of the each computation is half of the original RCS schemes, hence $r = 2$. The coded computations contain one submatrix from the first group, one submatrix from the second group, 3 submatrices from each group and 5 submatrices from each group, respectively[2]. For the given code structure we measure the average iteration time as 0.121, 0.087 and 0.075 for $q = 0, 0.15, 0.3$, respectively. One can observe from the results that by using smaller tasks both computation time and the computation load can be reduced. However, on the other hand use of smaller subtasks may increase the number of messages, for instance, given simulation setup the average number of received messages at the PS are 98, 80 and 70 for $q = 0, 0.15, 0.3$, respectively.

Although the RCS code is initially designed for coded computation, as explained in Section V, can be implemented for coded communication as well. Therefore, we repeat our experiments to compare the performances of RCS, UC-MMC and GC for a computational load of $r = 6$. For the RCS we now use the degree vector $\mathbf{d} = [1, 2, 3]$. The simulation results are presented in Table VII. We observe that in terms of the average per-iteration completion time UC-MMC and RCS both outperform GC, with UC-MMC typically achieving the lowest computation time. In terms of the communication load, UC-MMC has the worst performance. Hence, the key advantage of RCS in the coded communication scenario is a better balance between the computation and communication latencies. At this point, we also want to highlight that UC-MMC can be considered as a special case of RCS, where the degrees of the all message are one. Overall, one can play with the degree vector to achieve different points on the trade-off between the communication and computation latencies.

C. Discussions

We have proposed a new code construction framework for straggler-aware coded computation/communication schemes, which provides the flexibility to trade-off the accuracy of computation with the computation latency and the communication load. While we have provided a specific code construction, several improvements and/or adaptations of this design are possible. Below, we briefly discuss some of the possible future extensions.

1) Double threshold scheme: One of the possible extensions is to use two thresholds to decide when to terminate an iteration. The reason behind the use of two thresholds strategy is that partial recovery strategy is efficient when the sacrifice from the computation accuracy provides a noticeable reduction in the latency. However, if all the workers are sufficiently fast, then partial recovery may not bring noticeable reduction in the latency but lose accuracy. To this end, after sending the latest model vector $\boldsymbol{\theta}_t$, the PS starts keeping time and collects messages from workers for a given fixed duration. Once the duration is completed, PS checks whether the requirement due to tolerance level is satisfied or not and if it is not satisfied then continues to receive messages from workers until it is satisfied.

2) Adaptive tolerance : In the case of iterative training of machine learning models, it is known that the update accuracy has different impacts on the convergence at different phases of the training process. Hence, the tolerance can be adjusted over time to obtain a better overall convergence result.

3) Memory enhanced updates: Again, when partial computations are used in the context of iterative optimization or training, the PS can benefit from the computations recovered in the previous iterations. In our simulations, at each iteration we use only the computations recovered in that iteration. Instead, it can utilize the results from previous iterations to compensate for the missing computation results in the current iteration, similar to the *momentum SGD* framework.

IX. Conclusions

In this paper, we have introduced the CCPR approach, and a particular code structure, called RCS, in order to provide an additional flexibility in seeking a balance between the per-iteration completion time, the computation accuracy, and the communication load in distributed computing. In particular, for a matrix-vector multiplication task, the RCS code can adaptively recover a portion of the element of the resultant vector. The proposed code construction is built upon the LT code structure, but requires additional optimization of the underlying degree distribution due to the correlation among the erasures of symbols in the code. We have also shown that the RCS code can also provide a similar flexibility in distributed computation of any arbitrary computation task that can be written as the summation of multiple partial computations. We have applied the proposed RCS code to iterative SGD in a linear regression problem. By conducting experiments using different tolerance values we showed that the RCS code can help to reduce the per-iteration completion time for a reasonable reduction in the update accuracy, which can be tolerated due to the iterative nature of the algorithm. We also showed that, compared to UC-MMC, which can also employ partial recovery, RCS requires, on average, less number of messages to complete an iteration, which means lower communication load.

[2]Corresponding vector $\mathbf{z} = [1, 2, 1, 1, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2]$.

References

[1] S. Li, S. M. M. Kalan, A. S. Avestimehr, and M. Soltanolkotabi, "Near-optimal straggler mitigation for distributed gradient methods," in *2018*

IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), May 2018, pp. 857–866.

[2] N. Ferdinand and S. C. Draper, "Anytime stochastic gradient descent: A time to hear from all the workers," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct 2018, pp. 552–559.

[3] M. Mohammadi Amiri and D. Gündüz, "Computation scheduling for distributed machine learning with straggling workers," *IEEE Transactions on Signal Processing*, vol. 67, no. 24, pp. 6270–6284, Dec 2019.

[4] A. Behrouzi-Far and E. Soljanin, "On the effect of task-to-worker assignment in distributed computing systems with stragglers," in *2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Oct 2018, pp. 560–566.

[5] J. Chen, R. Monga, S. Bengio, and R. Józefowicz, "Revisiting distributed synchronous SGD," *CoRR*, vol. abs/1604.00981, 2016. [Online]. Available: http://arxiv.org/abs/1604.00981

[6] M. F. Aktaş and E. Soljanin, "Straggler mitigation at scale," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2266–2279, 2019.

[7] D. Wang, G. Joshi, and G. W. Wornell, "Efficient straggler replication in large-scale parallel computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, no. 2, pp. 7:1–7:23, Apr. 2019. [Online]. Available: http://doi.acm.org/10.1145/3310336

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

[9] M. Ye and E. Abbe, "Communication-computation efficient gradient coding," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. Stockholmsmässan, Stockholm Sweden: PMLR, 10–15 Jul 2018, pp. 5610–5619.

[10] W. Halbawi, N. Azizan, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," in *2018 IEEE Int. Symp. on Inf. Theory (ISIT)*, June 2018, pp. 2027–2031.

[11] E. Ozfatura, D. Gündüz, and S. Ulukus, "Gradient coding with clustering and multi-message communication," in *2019 IEEE Data Science Workshop (DSW)*, June 2019, pp. 42–46.

[12] S. Sasi, V. Lalitha, V. Aggarwal, and B. S. Rajan, "Straggler mitigation with tiered gradient codes," *IEEE Transactions on Communications*, pp. 1–1, 2020.

[13] L. Tauz and L. Dolecek, "Multi-message gradient coding for utilizing non-persistent stragglers," in *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, 2019, pp. 2154–2159.

[14] N. Charalambides, M. Pilanci, and A. O. Hero, "Weighted gradient coding with leverage score sampling," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 5215–5219.

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

[16] N. Ferdinand and S. C. Draper, "Hierarchical coded computation," in *2018 IEEE Int. Symp. Inf. Theory (ISIT)*, June 2018, pp. 1620–1624.

[17] R. K. Maity, A. Singh Rawa, and A. Mazumdar, "Robust gradient descent via moment encoding and ldpc codes," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2734–2738.

[18] S. Li, S. M. M. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *CoRR*, vol. abs/1805.09934, 2018.

[19] E. Ozfatura, D. Gündüz, and S. Ulukus, "Speeding up distributed gradient descent by utilizing non-persistent stragglers," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2729–2733.

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

[21] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4403–4413.

[22] H. Park, K. Lee, J. Sohn, C. Suh, and J. Moon, "Hierarchical coding for distributed computing," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1630–1634.

[23] A. Mallick, M. Chaudhari, U. Sheth, G. Palanikumar, and G. Joshi, "Rateless codes for near-perfect load balancing in distributed matrix-vector multiplication," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3366706

[24] S. Kiani, N. Ferdinand, and S. C. Draper, "Exploitation of stragglers in coded computation," in *2018 IEEE International Symposium on Information Theory (ISIT)*, June 2018, pp. 1988–1992.

[25] A. B. Das, L. Tang, and A. Ramamoorthy, "C3les: Codes for coded computation that leverage stragglers," in *2018 IEEE Information Theory Workshop (ITW)*, Nov 2018, pp. 1–5.

[26] E. Ozfatura, S. Ulukus, and D. Gündüz, "Distributed gradient descent with coded partial gradient computations," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 3492–3496.

[27] F. Haddadpour, Y. Yang, M. Chaudhari, V. R. Cadambe, and P. Grover, "Straggler-resilient and communication-efficient distributed iterative linear solver," *CoRR*, vol. abs/1806.06140, 2018.

[28] H. Wang, S. Guo, B. Tang, R. Li, and C. Li, "Heterogeneity-aware gradient coding for straggler tolerance," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, 2019, pp. 555–564.

[29] M. Kim, J. Sohn, and J. Moon, "Coded matrix multiplication on a group-based model," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 722–726.

[30] Y. Yang, M. Interlandi, P. Grover, S. Kar, S. Amizadeh, and M. Weimer, "Coded elastic computing," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 2654–2658.

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

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

[33] P. Soto, J. Li, and X. Fan, "Dual entangled polynomial code: Three-dimensional coding for distributed matrix multiplication," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 5937–5945.

[34] H. Park and J. Moon, "Irregular product coded computation for high-dimensional matrix multiplication," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 1782–1786.

[35] Y. Sun, J. Zhao, S. Zhou, and D. Gunduz, "Heterogeneous coded computation across heterogeneous workers," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[36] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Timely-throughput optimal coded computing over cloud networks," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. Mobihoc '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 301–310.

[37] B. Buyukates and S. Ulukus, "Timely distributed computation with stragglers," *IEEE Transactions on Communications*, vol. 68, no. 9, pp. 5273–5282, 2020.

[38] B. Hasırcıoğlu, J. Gómez-Vilardebó, and D. Gündüz, "Bivariate polynomial coding for efficient distributed matrix multiplication," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 814–829, 2021.

[39] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Private and rateless adaptive coded matrix-vector multiplication," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–25, 2021.

[40] R. Bitar, M. Wootters, and S. El Rouayheb, "Stochastic gradient coding for straggler mitigation in distributed learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 277–291, 2020.

[41] H. Wang, Z. B. Charles, and D. S. Papailiopoulos, "Erasurehead: Distributed gradient descent without delays using approximate gradient coding," *CoRR*, vol. abs/1901.09671, 2019. [Online]. Available: http://arxiv.org/abs/1901.09671

[42] S. Wang, J. Liu, and N. Shroff, "Fundamental limits of approximate gradient coding," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, Dec. 2019. [Online]. Available: https://doi.org/10.1145/3366700

[43] E. Ozfatura, S. Ulukus, and D. Gunduz, "Straggler-aware distributed learning: Communication computation latency trade-off," *Entropy, special issue Interplay between Storage, Computing, and Communications*

from an Information-Theoretic Perspective, vol. 22, no. 5, p. 544, May 2020.

[44] S. L. Smith, P.-J. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B1Yy1BxCZ

[45] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/ac1dd209cbcc5e5d1c6e28598e8cbbe8-Paper.pdf

[46] E. Ozfatura, B. Buyukates, D. Gündüz, and S. Ulukus, "Age-based coded computation for bias reduction in distributed learning," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[47] M. Luby, "LT codes," in *43rd Annual IEEE Symposium on Foundations of Computer Science*, November 2002.

[48] V. Bioglio, M. Grangetto, R. Gaeta, and M. Sereno, "An optimal partial decoding algorithm for rateless codes," in *2011 IEEE International Symposium on Information Theory Proceedings*, July 2011, pp. 2731–2735.

[49] S. Horii, T. Yoshida, M. Kobayashi, and T. Matsushima, "Distributed stochastic gradient descent using ldgm codes," in *2019 IEEE International Symposium on Information Theory (ISIT)*, July 2019, pp. 1417–1421.

[50] J. H. S. rensen, P. Popovski, and J. Ostergaard, "Design and analysis of lt codes with decreasing ripple size," *IEEE Transactions on Communications*, vol. 60, no. 11, pp. 3191–3197, 2012.

Emre Ozfatura received his B.Sc. in Electronics Engineering with Math minor and M.Sc. in Electronics Engineering from Sabanci University (Turkey), in 2012 and 2015, respectively. He is currently pursuing his Ph.D. degree at Imperial College London, UK, where he is a member of the Information Processing and Communications (IPC) Lab. His research interests are video streaming applications, distributed content storage and distributed computation.

Sennur Ulukus (Fellow, IEEE) is the Anthony Ephremides Professor in Information Sciences and Systems in the Department of Electrical and Computer Engineering at the University of Maryland at College Park, where she also holds a joint appointment with the Institute for Systems Research (ISR). Prior to joining UMD, she was a Senior Technical Staff Member at AT&T Labs-Research. She received the Ph.D. degree in Electrical and Computer Engineering from Wireless Information Network Laboratory (WINLAB), Rutgers University, and the B.S. and M.S. degrees in Electrical and Electronics Engineering from Bilkent University. Her research interests are in information theory, wireless communications, machine learning, signal processing and networks; with recent focus on private information retrieval, age of information, machine learning for wireless, distributed coded computing, group testing, physical layer security, energy harvesting communications, and wireless energy and information transfer.

Dr. Ulukus is a fellow of the IEEE, and a Distinguished Scholar-Teacher of the University of Maryland. She received the 2003 IEEE Marconi Prize Paper Award in Wireless Communications, the 2019 IEEE Communications Society Best Tutorial Paper Award, the 2020 IEEE Communications Society Women in Communications Engineering (WICE) Outstanding Achievement Award, the 2020 IEEE Communications Society Technical Committee on Green Communications and Computing (TCGCC) Distinguished Technical Achievement Recognition Award, a 2005 NSF CAREER Award, the 2011 ISR Outstanding Systems Engineering Faculty Award, and the 2012 ECE George Corcoran Outstanding Teaching Award. She was a Distinguished Lecturer of the IEEE Information Theory Society for 2018-2019.

She is an Area Editor for the IEEE Transactions on Wireless Communications (2019-present) and a Senior Editor for the IEEE Transactions on Green Communications and Networking (2020-present). She was an Area Editor for the IEEE Transactions on Green Communications and Networking (2016-2020), an Editor for the IEEE Journal on Selected Areas in Communications-Series on Green Communications and Networking (2015-2016), an Associate Editor for the IEEE Transactions on Information Theory (2007-2010), and an Editor for the IEEE Transactions on Communications (2003-2007). She was a Guest Editor for the IEEE Journal on Selected Areas in Communications (2008, 2015 and 2021), Journal of Communications and Networks (2012), and the IEEE Transactions on Information Theory (2011). She is the TPC chair of 2021 IEEE Globecom, and was a TPC co-chair of 2019 IEEE ITW, 2017 IEEE ISIT, 2016 IEEE Globecom, 2014 IEEE PIMRC, and 2011 IEEE CTW.

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

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