Straggler Mitigation Through Unequal Error Protection for Distributed Approximate Matrix Multiplication

Busra Tegin, *Graduate Student Member, IEEE*, Eduin E. Hernandez, Stefano Rini, and Tolga M. Duman, *Fellow, IEEE*

***Abstract*— Large-scale machine learning and data mining methods routinely distribute computations across multiple agents to parallelize processing. The time required for the computations at the agents is affected by the availability of local resources and/or poor channel conditions, thus giving rise to the "straggler problem." In this paper, we address this problem for distributed approximate matrix multiplication. In particular, we employ Unequal Error Protection (UEP) codes to obtain an approximation of the matrix product to provide higher protection for the blocks with a higher effect on the multiplication outcome. We characterize the performance of the proposed approach from a theoretical perspective by bounding the expected reconstruction error for matrices with uncorrelated entries. We also apply the proposed coding strategy to the computation of the back-propagation step in the training of a Deep Neural Network (DNN) for an image classification task in the evaluation of the gradients. Our numerical experiments show that it is indeed possible to obtain significant improvements in the overall time required to achieve DNN training convergence by producing approximation of matrix products using UEP codes in the presence of stragglers.**

***Index Terms*— Distributed computation, approximate matrix multiplication, stragglers, unequal error protection.**

I. Introduction

DISTRIBUTED computing is a fundamental approach to the training of machine learning models as it allows for parallel computation of model updates. Parallelizing computation enhances robustness, reliability, and allows for a drastic reduction in computational and memory resource requirements at the learner. Distributed computation is supported by a dedicated infrastructure, often comprised of computing clusters with heterogeneous capabilities. The widespread reliance on distributed computation clusters presents several opportunities over traditional computing paradigms, but also offers a new set of challenges. Among the most well-recognized issues is that of the stochasticity in the time required for the computation. This gives rise to the phenomenon of "stragglers," that is, agents with large response times which delay computation. Another important reason for delayed responses from the computation nodes is due to the wireless channel effects. Workers observe different channel conditions, which result in lower transmission rates and longer transmission times for the ones with poor quality links. This phenomenon is particularly relevant for edge computing, in which users with high-mobility rely on a large network of computation nodes with varying wireless and wired connection capabilities. As a remedy to stragglers, channel coding can be applied to reduce the delays in distributed computation [2].

In this paper, we propose a novel scheme for distributed approximate matrix computation in the presence of stragglers which makes use of the variations in the magnitude of the matrix entries to alleviate the effects of computation delays. Such variations in the magnitudes of matrix entries naturally occur in many applications such as gradient computation for back-propagation in Deep Neural Network (DNN) training. We first identify the matrix sub-products which are expected to have the largest norms and use Unequal Error Protection (UEP) coding to provide robustness against stragglers. The proposed solution offers an improved resilience by providing better approximate reconstruction of the matrix product by a given deadline.

A. Literature Review

As matrix multiplication is a fundamental algebraic operation, distributed approximate matrix multiplication has been investigated in many contexts. In the big-data paradigm, computation and storage are distributed; hence computer processing architectures can be devised to efficiently perform this operation [3], [4]. In a cloud-computing setting, distributed matrix computation is investigated in [5], [6]. DNN training through back-propagation involves multiplication of large matrices, for which distributed matrix computation is studied in [7], [8]. More recently, the problem of distributed matrix multiplication in the presence of stragglers has been

Manuscript received March 1, 2021; revised September 13, 2021; accepted September 22, 2021. Date of publication October 6, 2021; date of current version January 17, 2022. This article was presented in part at the 2021 IEEE International Conference on Communications (ICC), Montreal, Canada, in June 2021 [DOI: 10.1109/ICC42927.2021.9500264]. *(Corresponding author: Busra Tegin.)*

Busra Tegin and Tolga M. Duman are with the Department of Electrical and Electronics Engineering, Bilkent University, 06800 Ankara, Turkey (e-mail: btegin@ee.bilkent.edu.tr; duman@ee.bilkent.edu.tr).

Eduin E. Hernandez and Stefano Rini are with the Department of Electrical and Computer Engineering, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: eduin.ee08@nycu.edu.tw; stefano.rini@nycu.edu.tw).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2021.3118350.

Digital Object Identifier 10.1109/JSAC.2021.3118350

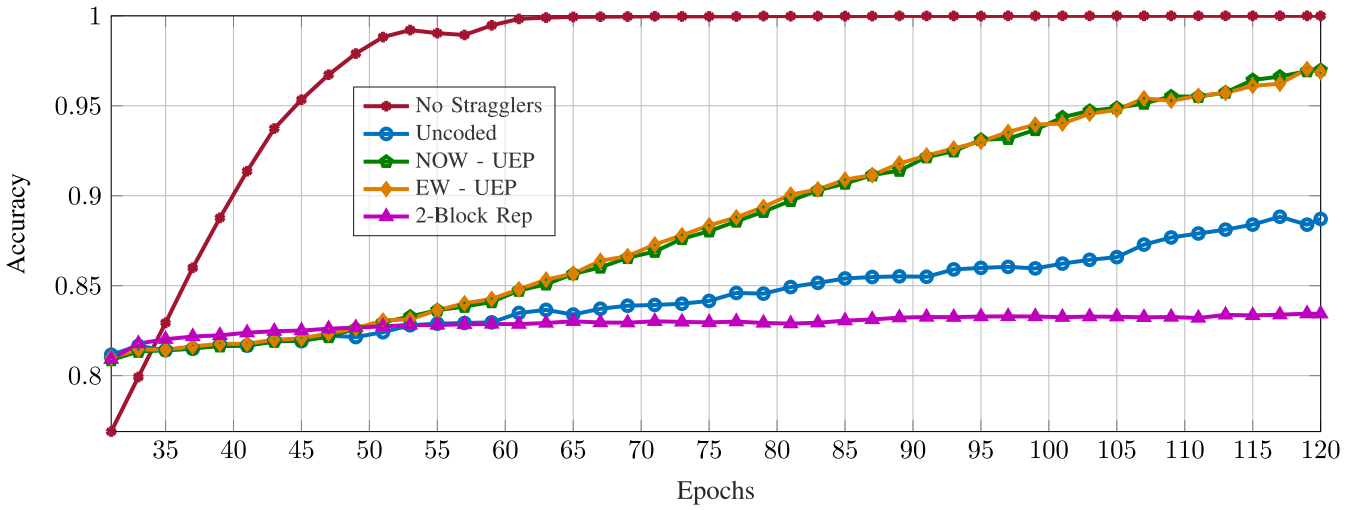


Fig. 1. CIFAR-10 classification accuracy between epoch 30 and epoch 120 with $\lambda = 0.5, T_{max} = 1$. Evaluation details are presented in Sec. VII-C.

considered. Channel coding can be utilized for matrix multiplication to mitigate the effect of stragglers [9]. Since its inception in [9], this line of research has received significant attention in the literature. In [10], the authors use the theory of extreme order statistics to analyze how task replication reduces latency. In [11], the authors introduce redundant computations in a coding theory-inspired fashion for computing linear transforms of long vectors. Product codes for distributed matrix multiplication are studied in [12]. A new class of codes, called polynomial codes, is proposed and their optimality is argued for the straggler problem in [13].

While the above literature focuses on minimizing the time for completing a computation task, one can also consider approximate computation as way to speed up computation and reduce the effects of stragglers. Along these lines, in [5], the authors propose OverSketch, an algorithm that uses matrix sketching to approximate matrix multiplication. Further research considers the intersection of distributed matrix computation and other relevant aspects of computation. For instance, the authors in [14] consider the distributed matrix multiplication problem when the usefulness of the computation outcome is evaluated through an age-of-information paradigm [15].

B. Contribution

In this paper, we investigate the trade-off between accuracy and delay in distributed approximate matrix multiplication with stragglers. Since for typical machine learning problems, only approximate matrix multiplication results are sufficient, we consider a distributed matrix multiplication scheme in which the sub-blocks of the matrices being multiplied are encoded using UEP codes and distributed across different workers. Due to, for instance, job queue status and wireless channel effects, the workers respond at random times, with the results of the products of the coded sub-blocks. The parameter server (PS) chooses the protection level of each matrix sub-block according to its norm so that the sub-products with the largest contribution suffer the least from the effects of stragglers. Our main goal is to produce an approximation of the product of two matrices as quickly as possible; with a more accurate approximation as more workers respond. This produces a progressively improving matrix approximation over time, thus fully exploiting the properties of UEP codes.

Our main contribution is employing UEP codes to improve the quality of the approximation of matrix multiplications by exploiting the variations in the matrix entries' magnitudes. In particular, we leverage the construction of UEP codes described in [16] through Random Linear Codes (RLC) to offer more protection to the sub-products with larger norms and reduce the effects of the randomness in the service time. Specifically, we consider two schemes: Non-Overlapping Windows (NOW) and Expanding Window (EW) RLC codes for UEP, and analyze the performance of the proposed approximate matrix multiplication schemes. Different from the existing literature, we consider two different partitioning schemes for the matrices to be multiplied: (i) row-times-column block products, and (ii) column-times-row block products which are encoded and distributed among a set of workers which can perform sub-matrix multiplications. To illustrate the importance of our proposed strategy for distributed machine learning algorithms, we construct a DNN training with CIFAR-10 and MNIST datasets, which are extensively used datasets when evaluating the performance of machine learning applications, in a scenario where multiplications in the back-propagation step are distributed among workers using the NOW-UEP and EW-UEP codes.

To showcase our results, the performance of our approach for this scenario is presented in Fig. 1 where the training performance attainable through our algorithm for the CIFAR-10 image classification database between 30 and 120 epochs are depicted. In this simulation, we let the response time of the servers be exponentially distributed with a mean inversely proportional to the number of sub-block multiplications, thus accounting for the larger number of tasks when employing coding. Three reference curves in the plot are the red curve, corresponding to the case with no stragglers, i.e., a deterministic response time; the blue curve, corresponding the perfor-

mance with uncoded transmission; and, the purple curve for which computations are simply replicated. The performance attainable through UEP codes for the approximate computation of the weight updates are depicted as green and yellow curves. The results clearly show that the UEP codes provide a higher model accuracy in the presence of stragglers. Further analysis and interpretation are provided in Sec. VII-C.

Organization: The paper is organized as follows. In Sec. II, we formulate the distributed approximate matrix multiplication problem for both (i) row-times-column block products and (ii) column-times-row block products. In Sec. III, we go over some of the existing results in the literature for coded matrix computation and approximate matrix multiplication. In Sec. IV, we present our proposed scheme in which UEP codes are used to encode the matrix multiplication factors, while a theoretical evaluation of the expected error is provided in Sec. V. In Secs. VI and VII, we provide numerical examples using both synthetic data and an actual data from DNN training. Finally, the paper is concluded in Sec. VIII.

Notation: Matrices are denoted with bold capital Roman letters, e.g., $\mathbf{A}$, column vectors with bold lower-case Roman letters, e.g., $\mathbf{v}$. The Frobenius norm of the matrix $\mathbf{A}$ is shown as $\|\mathbf{A}\|_F$. The set of integers $\{1,\dots,N\} \subset \mathbb{N}$ is denoted as $[N]$. Given two matrices $\mathbf{A}_1$ and $\mathbf{A}_2$ with the same number of rows, we depict their column-wise concatenation as $\mathbf{A} = [\mathbf{A}_1\,,\,\mathbf{A}_2]$. Similarly, given $\mathbf{A}_1$ and $\mathbf{A}_2$ with the same number of columns, their row-wise concatenation is represented as $\mathbf{A} = [\mathbf{A}_1\,;\,\mathbf{A}_2]$ which can also be equivalently expressed as $\mathbf{A} = [\mathbf{A}_1^\intercal\,,\,\mathbf{A}_2^\intercal]^\intercal$. Capital Roman letters are used for scalars. $\mathcal{N}(\mu,\sigma^2)$ indicates the Gaussian distribution with mean μ and variance σ^2. Finally, expectation is denoted as $\mathbb{E}[\cdot]$, and $\mathbf{1}(\cdot)$ is used for the indicator function.

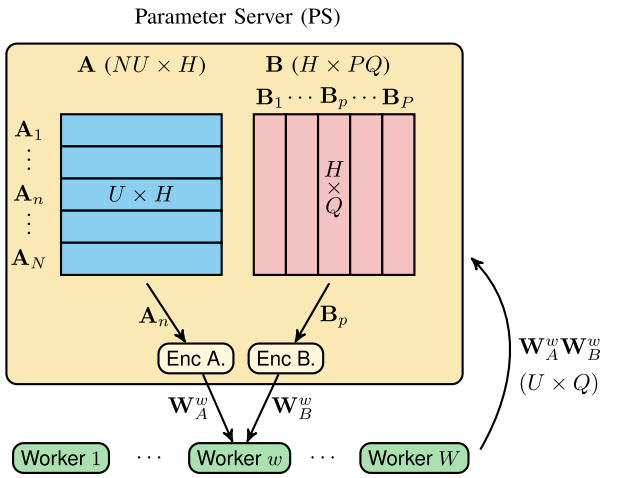
Note well: In the following, we will often not explicitly indicate the support of the independent variables indexing the various matrix sub-blocks. We shall use lower case Roman letters for such independent variables, i.e., n, and let the corresponding upper case Roman letter indicate the interval $n \in [N]$, in other words

$$\sum_{n\in[N]} f_n \triangleq \sum_{n} f_n.$$

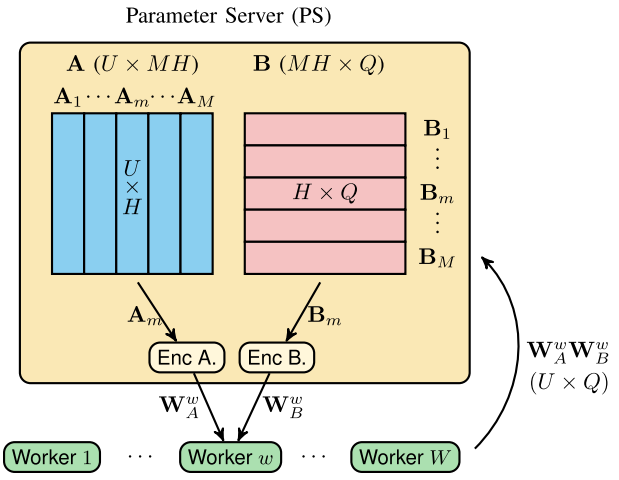
II. System Model

We consider the scenarios in Fig. 2 where a PS wishes to compute the matrix product $\mathbf{C} = \mathbf{AB}$ by distributing various factors of the matrix multiplication among W workers. Each worker receives two separate linear combinations of sub-matrices of $\mathbf{A}$ and $\mathbf{B}$, computes their product, and returns it to the PS. The time required at the worker for responding to a computation request from the server is a random variable due to various factors, including (but not limited to) variations in the channel quality, worker queue status, and heterogeneity in the worker's computational speeds [17]–[19].

We assume that the server distributes the same amount of computation to all the workers. By a given deadline, the PS produces an approximation $\widehat{\mathbf{C}}$ of the matrix $\mathbf{C}$ by using the sub-products received from the workers by the prescribed deadline. We consider the problem of designing an algorithm that distributes the matrix computation to the prescribed number of workers to reconstruct an approximation of the desired matrix product within a prescribed precision, as measured by the Frobenius norm.



(a) Row-times-column (r × c) multiplication.



(b) Column-times-row (c × r) multiplication.

Fig. 2. System models with the r × c and c × r multiplication schemes.

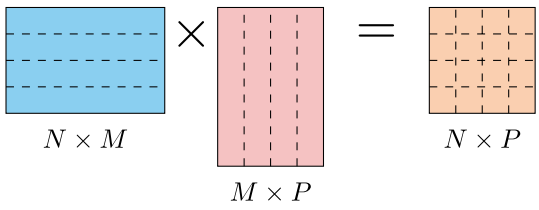
A. Distributed Matrix Computation Model

Consider the matrices $\mathbf{A}$ and $\mathbf{B}$ with elements from a finite field $\mathbb{F}$. The matrix $\mathbf{A}$ is comprised of $N \times M$ sub-blocks of dimensions $U \times H$, thus resulting in the overall dimensions $NU \times MH$. Similarly, $\mathbf{B}$ is comprised of $M \times P$ sub-blocks of dimensions $H \times Q$ resulting in $MH \times PQ$. Accordingly, the matrix $\mathbf{C}$ has $N \times P$ sub-blocks of dimension $U \times Q$. Thus, $\mathbf{A} \in \mathbb{F}^{NU\times MH}$, $\mathbf{B} \in \mathbb{F}^{MH\times PQ}$, and $\mathbf{C} \in \mathbb{F}^{NU\times PQ}$.

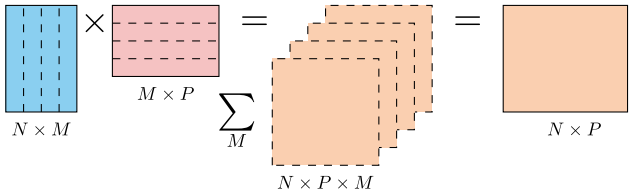
The aim of the PS is to produce $\widehat{\mathbf{C}}$ as an approximate expression for the matrix multiplication $\mathbf{C} = \mathbf{AB}$ with respect to the loss[1]

$$\mathcal{L}(\mathbf{C},\widehat{\mathbf{C}}) = \|\mathbf{C} - \widehat{\mathbf{C}}\|_F^2. \tag{1}$$

[1]In the following, we only consider the case of a Frobenius norm: the case of a more general loss is not discussed for brevity.



(a) Row-times-column multiplication ($\mathsf{r} \times \mathsf{c}$) in (2).



(b) Column-times-row multiplication ($\mathsf{c} \times \mathsf{r}$) in (3).

Fig. 3. A conceptual representation of the $\mathsf{r} \times \mathsf{c}$ and $\mathsf{c} \times \mathsf{r}$ multiplication partitioning.

To accomplish this, the PS divides the matrix products into sub-products and distributes them across a set of workers. Specifically, following [20], we partition $\mathbf{A}$ and $\mathbf{B}$ in two possible ways:

- **Row-times-Column** ($\mathsf{r} \times \mathsf{c}$)**:** that is, $M = 1$ and $H = MH$ such that H has the same number of columns as A and number of rows as B.
  In this case, if uncoded multiplication is distributed to the servers, what is returned are NP sub-blocks of sizes $U \times Q$ belonging to the $\mathbf{C}$ matrix, so that
  $$\mathbf{A} = [\mathbf{A}_1 \, ; \cdots \mathbf{A}_n \, ; \cdots \mathbf{A}_N],$$
  $$\mathbf{B} = [\mathbf{B}_1 \, , \cdots \mathbf{B}_p \, , \cdots \mathbf{B}_P]. \tag{2}$$
  We indicate this case with the notation $\mathsf{r} \times \mathsf{c}$. This partitioning is presented in Figs. 2a and 3a.
- **Column-times-Row** ($\mathsf{c} \times \mathsf{r}$)**:** that is, $N = P = 1$, $U = NU$, $Q = PQ$ such that U has the same number of rows as A and Q has the same number of columns as B. In this case, if uncoded multiplication is distributed to the servers, what is returned are M matrices of sizes $U \times Q$, so that
  $$\mathbf{A} = [\mathbf{A}_1 \, , \cdots \mathbf{A}_m \, , \cdots \mathbf{A}_M],$$
  $$\mathbf{B} = [\mathbf{B}_1 \, ; \cdots \mathbf{B}_m \, ; \cdots \mathbf{B}_M]. \tag{3}$$
  We indicate this case with the notation $\mathsf{c} \times \mathsf{r}$. This partitioning is presented in Figs. 2b and 3b.

In general, not all the sub-blocks have the same impact on the final matrix multiplication result, as some sub-blocks may have larger Frobenius norms than others. This motivates the use of codes to efficiently trade-off the matrix approximation with the computation delay. In other words, codes can be employed to better protect the more impactful sub-products when distributing the computation to the workers, so that a more precise approximation is produced in a shorter time. For this reason, we consider the coding problem in which the PS sends the matrices $\mathbf{W}_A^w$ and $\mathbf{W}_B^w$ obtained as

$$\mathbf{W}_A^w = \begin{cases} f_{\text{enc}-A}\left(\mathbf{A}_1, \cdots, \mathbf{A}_N\right) & \mathsf{r} \times \mathsf{c} \\ f_{\text{enc}-A}\left(\mathbf{A}_1, \cdots, \mathbf{A}_M\right) & \mathsf{c} \times \mathsf{r}, \end{cases} \tag{4}$$

$$\mathbf{W}_B^w = \begin{cases} f_{\text{enc}-B}\left(\mathbf{B}_1, \cdots, \mathbf{B}_P\right) & \mathsf{r} \times \mathsf{c} \\ f_{\text{enc}-B}\left(\mathbf{B}_1, \cdots, \mathbf{B}_M\right) & \mathsf{c} \times \mathsf{r}, \end{cases} \tag{5}$$

to each worker w and sets a time deadline $T_{\max}$ by which it expects the matrix products $\{\mathbf{W}_A^w \mathbf{W}_B^w\}_{w \in [W]}$ to be returned where $f_{\text{enc}-A}$ and $f_{\text{enc}-B}$ are the encoding functions for the sub-matrices of $\mathbf{A}$ and $\mathbf{B}$, respectively. At time $T_{\max}$, the PS produces the approximation of the matrix product $\mathbf{C} = \mathbf{AB}$ as

$$\widehat{\mathbf{C}} = f_{\text{dec}-C}\left(\mathcal{W}(T_{\max})\right), \tag{6}$$

where $\mathcal{W}(T_{\max}) \subseteq \{\mathbf{W}_A^w \mathbf{W}_B^w\}_{w \in [W]}$ is the set of matrix products received up to time $T_{\max}$ where $f_{\text{dec}-C}$ is the decoding function for the final product estimate $\widehat{\mathbf{C}}$. Using $\widehat{\mathbf{C}}$ in (6), the loss in (1) can be evaluated: let us denote it as $\mathcal{L}(T_{\max})$.

Note that the set $\mathcal{W}(T_{\max})$ is random, which follows from the randomness of the response times. More precisely, we assume that the response time of the workers are random variables denoted by T_w which are identical and independently distributed (i.i.d.) with a cumulative distribution function (CDF)

$$F_{T_w}(t) = F(t), \quad w \in [W]. \tag{7}$$

This CDF is used to model the response times of the workers due to the transmission rate constraints of the wireless channel, computational delays, and network-level transmission delays either between the server and the workers or vice-versa.

The problem we consider in the following is to design the functions in (4), (5) and (6) in such a way that the loss in (1), averaged over the randomness in $\mathcal{W}(T_{\max})$, is minimized over some dataset of matrix multiplications $\mathcal{D}(\{\mathbf{A}, \mathbf{B}\})$. Let us define this quantity as a function of the waiting time $T_{\max}$ as

$$\mathsf{L}(T_{\max}) = \min \; \mathbb{E}\left[\sum_{\mathcal{D}} \mathcal{L}(\mathbf{C}, \widehat{\mathbf{C}})\right], \tag{8}$$

where (i) the minimization in (8) is over $f_{\text{enc}-A}$,$f_{\text{enc}-B}$ and $f_{\text{dec}-C}$, (ii) the expectation is over $\mathcal{W}(T_{\max})$, (iii) the summation is over all matrices in the database $\mathcal{D}$, and all the matrices in the database are equally likely.

Remark 1 (Comparison Across Models): In the following, we will be interested in comparing the performance when the number of workers varies. For this comparison, we will consider the scaling of the response times as in (7) *as* $F(\Omega t)$*, where* Ω *is the number of matrix sub-products divided by the number of workers.*

Remark 2 (Matrix Partitioning Paradigms): A representation of the $\mathsf{r} \times \mathsf{c}$ *paradigm is provided in Fig. 2a: in Fig. 3a we represent the resulting block-matrix structure of* $\mathbf{C}$*. We observe that each sub-product in* $\mathcal{W}(T_{\max})$ *contributes one sub-block in* $\mathbf{C}$*. The* $\mathsf{c} \times \mathsf{r}$ *paradigm is represented in Fig. 2b: in this*

TABLE I
A SUMMARY OF THE QUANTITIES IN SEC. II (COLUMN II AND III) AND SEC. IV (COLUMN IV AND V)

Multiplication Case	Matrix	Size	Constant	Value
General	$\mathbf{A}$	$NU \times MH$	# of workers	W
	$\mathbf{B}$	$MH \times PQ$	# of importance levels ($\mathbf{A}/\mathbf{B}$)	S
	$\mathbf{C}$	$NU \times PQ$	# of importance levels ($\mathbf{C}$)	L
$\mathsf{r} \times \mathsf{c}$	$\mathbf{A}_n$	$U \times H$	# of row blocks ($\mathbf{A}$)	N
	$\mathbf{B}_p$	$H \times Q$	# of column blocks ($\mathbf{B}$)	P
	$\mathbf{C}_{np}$	$U \times Q$	Response time scaling	Ω
$\mathsf{c} \times \mathsf{r}$	$\mathbf{A}_m$	$U \times H$	# of column blocks ($\mathbf{A}$)	M
	$\mathbf{B}_m$	$H \times Q$	# of row blocks ($\mathbf{B}$)	M
	$\mathbf{C}_m$	$U \times Q$	Deadline	$T_{\max}$

paradigm $\mathbf{C}$ *is obtained as a sum of rank-one terms, or outer products, as shown in Fig. 3b. In this case, each sub-product in* $\mathcal{W}(T_{\max})$ *contributes to one of such rank-one terms.*

Remark 3: Let us elaborate further on the notion used in (2) *and* (3)*. Stated more precisely, the sub-block matrix structure of* $\mathbf{A}, \mathbf{B}$ *and* $\mathbf{C}$ *is as follows: The matrix* $\mathbf{A}$ *is comprised of* NM *sub-matrices* $\mathbf{A}_{nm}$ *with* $n \in [N]$, $m \in [M]$, *and* $\mathbf{A}_{nm} \in \mathbb{F}^{U \times H}$. *Similarly, the matrix* $\mathbf{B}$ *is comprised of* MP *sub-matrices* $\mathbf{B}_{mp}$ *with* $m \in [M]$, $p \in [P]$, *and* $\mathbf{B}_{mp} \in \mathbb{F}^{H \times Q}$. *Accordingly, the matrix* $\mathbf{C}$ *is composed of* NP *sub-matrices* $\mathbf{C}_{np}$ *for* $n \in [N]$, $p \in [P]$, *and with* $\mathbf{C}_{np} \in \mathbb{F}^{U \times Q}$. *Note that here* NP *block-matrix multiplications are needed to produce* $\mathbf{C}$.

- $\mathsf{r} \times \mathsf{c}$ *scenario: For the row-times-column case in* (2)*, we have that* $\mathbf{A}_n \in \mathbb{F}^{U \times H}$ *and* $\mathbf{B}_p \in \mathbb{F}^{H \times Q}$ *for* $n \in [N]$, *and* $p \in [P]$. $\mathbf{C}_{np} \in \mathbb{F}^{U \times Q}$ *and* NP *such block-matrix multiplications are needed to produce* $\mathbf{C}$.
- $\mathsf{c} \times \mathsf{r}$ *scenario: For the column-times-row case in* (3)*, we have* $\mathbf{A}_m \in \mathbb{F}^{U \times H}$ *and* $\mathbf{B}_m \in \mathbb{F}^{H \times Q}$ *for* $m \in [M]$. $\mathbf{C}_m \in \mathbb{F}^{U \times Q}$ *and* M *such block matrix multiplications and summations are needed to produce* $\mathbf{C}$. *Note that the notation* $\mathsf{r} \times \mathsf{c}$ / $\mathsf{c} \times \mathsf{r}$ *for the row-times-column case/column-times-row indicates that* $M = 1$ / $N = P = 1$, *respectively.*

A summary of the notation introduced in this section is provided in Table I.

B. Deep Learning Motivation

Let us now briefly motivate the choice of the system model in Sec. II-A in the context of distributed training of DNN. Note that we will further comment on this application of our results in Sec. VII-C.

Generally speaking, we observe that the matrices involved in the back-propagation, both weight and gradient matrices, have a sparse nature, which is often also enforced through sparsification techniques. It can also be observed that the sparsity level varies across DNN layers, often deeper layers being sparser than shallower ones. The presence of sparsity in these matrices means that the UEP codes have the potential to drastically improve the back-propagation speed by utilizing approximate matrix multiplication.

In DNN training, sparsity is often explicitly introduced in order to provide robustness or to reduce the communication load of the training process. One of the most straightforward sparsification approaches is to set all the DNN weights below a certain threshold to zero. These threshold values are increased as the training progresses, so as to enforce sparsity in the final weights. In Fig. 4 we plot a Gaussian fitting of the gradients, weights, and inputs at different layers for a DNN model trained over the MNIST dataset for digit classification. Since the inputs have gone through Rectified Linear Units (ReLU), these values are nonnegative. In these simulations, we appropriately choose a sparsity level and report it in Table II. Note that at least half of the matrix entries are well approximated by zero entries. As it can be observed from Fig. 4a, the non-sparse entries can be well described as Gaussian random variables with a mean close to zero. In Figs. 4b and 4c, we plot a Gaussian fitting of the gradient and weight values, showing that the non-sparse matrix entries are roughly Gaussian distributed with a variance increasing with the layer depth. Finally, as the inputs of some DNN layers have gone through a ReLU activation function, these matrix entries are roughly half Gaussian distributed.

TABLE II
SPARSENESS OF THE MATRICES MODELED IN FIG. 4

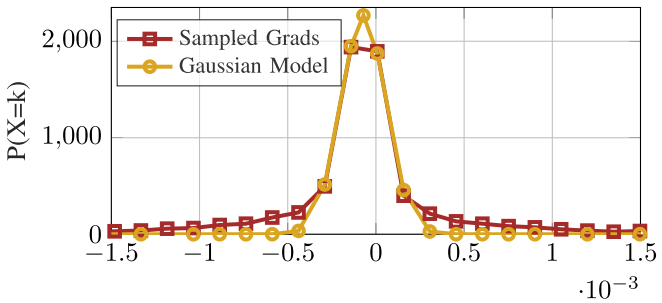
Layer	Gradients	Weight	Input
1	50.09%	0.15%	-
2	59.09%	0.11%	33.11%
3	57.97%	0.10%	38.63%

III. RELEVANT RESULTS

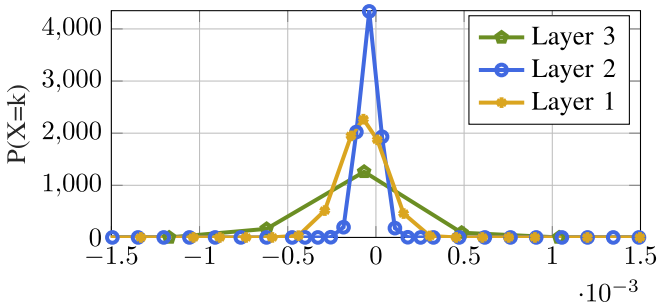
A. Approximate Matrix Computation

In many applications, one is interested in the trade-off between the time required for computation and the precision of the computation. For instance, consider the scenario in which the PS is training a DNN so that the matrix multiplication corresponds to the gradient back-propagation operation. In this scenario, an approximate evaluation of the matrix product results in an overall improvement of the training time as the speed-up benefits far out-weigh the loss of precision in the computation results.

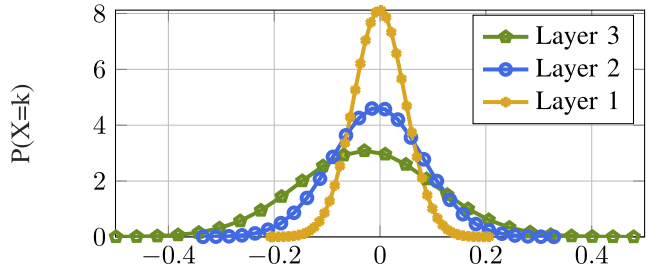
Approximate matrix multiplication has a long history in mathematics, computer science, and engineering. The problem was initially considered in [21], inspired by the problem of finding low-rank matrix approximations in [22]. When



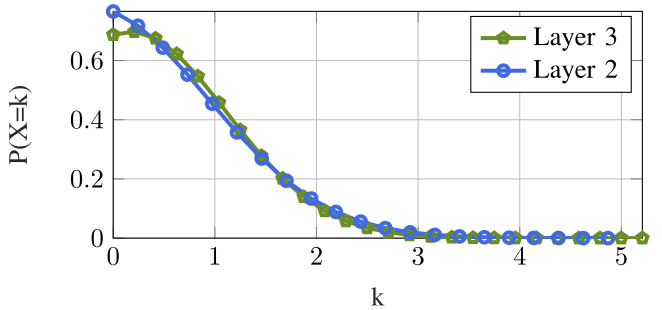
(a) Gaussian fitting of the sampled gradient from layer 1.



(b) Gradient fitting:
Layer 1 $\approx \mathcal{N}(-7.09\text{E}-05, 7.24\text{E}-01)$
Layer 2 $\approx \mathcal{N}(-3.90\text{E}-05, 6.31\text{E}-01)$
Layer 3 $\approx \mathcal{N}(-1.02\text{E}-04, 2.56\text{E}-04)$



(c) Weight fitting:
Layer 1 $\approx \mathcal{N}(-1.07\text{E}-03, 9.99\text{E}-01)$
Layer 2 $\approx \mathcal{N}(-4.40\text{E}-03, 1.00\text{E}+00)$
Layer 3 $\approx \mathcal{N}(-2.71\text{E}-02, 9.98\text{E}-01)$



(d) Input fitting:
Layer 2 $\approx ReLU(\mathcal{N}(-2.40\text{E}-01, 2.28\text{E}+00))$
Layer 3 $\approx ReLU(\mathcal{N}(1.69\text{E}-01, 1.66\text{E}+00))$

Fig. 4. Gaussian modeling for the dense portion of each layer for the MNIST classification task at mini-batch iteration 389 / 937 with the DNN model defined in Fig. 11 and parameters in Table IV.

considering a simple algorithm in which the matrix multiplication is approximated by randomly sampling columns from $\mathbf{A}$ and rows from $\mathbf{B}$ (i.e., an outer product) and accumulating these rank-one matrices to produce an approximation of $\mathbf{C}$, an approximation loss of

$$\mathcal{L}(\mathbf{C}, \widehat{\mathbf{C}}) = \mathcal{O}\left(\frac{\|\mathbf{A}\|_F \|\mathbf{B}\|_F}{\sqrt{c}}\right), \tag{9}$$

is obtained where c is the number of outer products accumulated to produce $\widehat{\mathbf{C}}$ [22]. A similar interpretation is also presented in [23] for block/submatix sampling.

The above approach is referred as random projection or "sketching." Many studies are based on the Johnson-Lindenstrauss (JL) Lemma [24]–[26]. In [27], the authors aim for increased speed-ups by introducing a sparse JL. In [28], the authors come up with a very sparse subspace embedding matrix with modifications where their results can be used to speed up approximate matrix multiplication by dimension reduction after the JL projection.

B. UEP Codes

The substantial reduction in complexity attained through approximate matrix multiplication discussed in Sec. III-A comes from identifying the matrix features which influence the multiplication outcome the most. In the following, we will rely on this observation and also choose the protection levels of our matrix products according to this influence. To do so, we rely on UEP codes.

UEP codes were introduced in [29], where the authors perform an analysis for linear codes to protect a set of digits at a higher level than other sets of digits and provide methods of synthesizing UEP codes using parity check matrices. Since then, there have been many studies which investigate UEP codes to provide unequal protection for different positions in a data sequence.

UEP codes are also studied for multimedia transmission schemes with network coding, see e.g., [30], [31]. These studies divide the source packet into several layers and importance levels to apply an RLC scheme which provides better protection for certain sub-packets for high priority recovery. In [16], the authors consider a similar setup; further, they also provide performance analysis for packet-level UEP coding over packet erasure channels.

While there are different ways of UEP coding, here we focus on the codes introduced in [16], which are also based on RLC. Two families of codes are analyzed in [16]: NOW-UEP and EW-UEP, respectively. In both codes, we consider the case of a layered message source which consists of K equal-length source packets of b bits each. Without loss of generality, assume that bits are ordered in decreasing levels of importance. A window containing a subset of these packets is then considered, and packets inside each window are selected according to a given probability. Once selected, a random linear code is employed to produce the parity bits.

In the NOW-UEP coding strategy, the selection windows are non-overlapping. The encoding of the information bits is performed by selecting a window using a window selection polynomial $\Gamma(\xi) = \sum_l \Gamma_l \xi^l$, where Γ_l is the window selection probability for the l-th importance level. Then, the encoded messages are generated only from the importance levels of the selected type, as illustrated in Fig. 5. With this coding strategy,

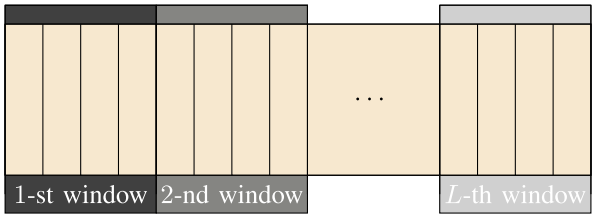


Fig. 5. Window definition for UEP-NOW codes in [16].

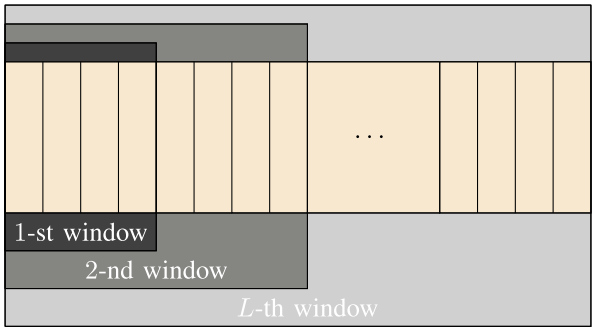


Fig. 6. Window definition with UEP-EW codes [16].

separate and independent coding is applied to each importance level enabling different rate allocation for different levels with the help of the window selection probabilities. Thus, one can interpret this coding scheme as applying different error protection codes for each level.

The EW-UEP coding strategy also uses a probabilistic window selection polynomial $\Gamma(\xi)$ for the l-th importance level; however, the window definition is different from that of the NOW-UEP strategy. The EW-UEP constructs the l-th window by including all the packets whose importance level is l or higher as illustrated in Fig. 6. Therefore, the packets in the first window are included in all the encoded messages; hence they are the best protected ones. The packets with lower importance are used less in the encoding process resulting in less protection for them. Therefore, progressive protection is provided for the source packets. For instance, let us assume that the third importance level is selected according to the window selection distribution, then the coded packet includes all the source messages from the first, second, and third importance levels. Thus, the EW strategy includes the most important matrices to the encoding process regardless of the importance level of the selected window to provide a better protection than the others.

It is also worth noting that encoding and decoding processes of RLC have a small complexity level which makes them feasible for real-time implementation. For instance, in [32], [33], it is demonstrated that real-time implementation of RLC over wireless channels is even possible by using smartphones. Hence, the overhead introduced by our approach can be neglected compared to the complexity of matrix multiplication.

IV. Approximate Matrix Multiplication Through UEP Codes

In this section, we propose a distributed coded approximate matrix multiplication scheme which aims to provide better protection for the matrix sub-products of $\mathbf{A}_{nm} \in \mathbb{R}^{U\times H}$ and $\mathbf{B}_{mp} \in \mathbb{R}^{H\times Q}$ with larger norms, and thus produce a better approximation of the matrix product within a prescribed deadline. In particular, the coding scheme relies on a parametrization of the protection levels matching the distribution of the Frobenius norms of the rows and columns of the matrices in question.

A. Importance Level of a Sub-Block

1) Importance Levels for $\mathsf{r}\times\mathsf{c}$ *Multiplication:* Let us begin by classifying the matrix sub-blocks in (2) according to their norms. For instance, we may select three different levels for each sub-block, e.g., *high*, *medium*, and *low* to classify the norms of $\mathbf{A}_n$ and $\mathbf{B}_p$. Let us refer to these levels as *importance levels*, and assume that there are S such levels and let the importance be decreasing in $s \in [S]$. Given a matrix $\mathbf{A/B}$, we have $n_A(s)/n_B(s)$ blocks with the importance level $s \in [S]$. Clearly, $N = \sum_s n_A(s)$, and $P = \sum_s n_B(s)$.

By construction, any sub-product $\mathbf{C}_{np}$ is obtained as the multiplication of sub-blocks in two classes: accordingly $\mathbf{C}_{np}$ has L possible importance levels with $L = S(S+1)/2$. For instance, for the example of three importance levels, for both $\mathbf{A}_n$ and $\mathbf{B}_p$, $\mathbf{C}_{np}$ can have importance *high* $\times$ *high*, *high* $\times$ *medium*, *high* $\times$ *low*, *medium* $\times$ *medium*, etc.

From a high level-perspective, one would want the PS to be able to quickly recover those products corresponding to the importance level *high* $\times$ *high*, while the importance level *low* $\times$ *low* is not particularly urgent. We can obtain this desired behavior by employing UEP codes.

2) Importance Levels for $\mathsf{c}\times\mathsf{r}$ *Multiplication:* Similar to the previous case, we will classify the matrix sub-blocks in (3) based on their norms into S importance levels. However, this time the classification will be based on the column sub-blocks of $\mathbf{A}$ and row sub-blocks of $\mathbf{B}$ which are denoted by $\mathbf{A}_m$ and $\mathbf{B}_m$, respectively. Similarly, we have $n_A(s)/n_B(s)$ for $s \in [S]$ as the number of blocks, and $M = \sum_s n_A(s) = \sum_s n_B(s)$.

As a result of the multiplication of $\mathbf{A}_m$ and $\mathbf{B}_m$, $\mathbf{C}_m$ will be comprised of L different importance levels which will depend on the pairing of column and row importance levels of $\mathbf{A}$ and $\mathbf{B}$, respectively. Note that, for $\mathsf{c}\times\mathsf{r}$ multiplication, we only have M sub-block products due to multiplication of $\mathbf{A}_m$ and $\mathbf{B}_m$, $m \in [M]$, which may result in less than $S(S+1)/2$ importance levels depending on the order of the sub-blocks. Different from the previously described block partitioning, each sub-block multiplication results in a matrix $\mathbf{C}_m$ whose size is same as the original multiplication $\mathbf{C} = \mathbf{AB}$. Note that, for perfect recovery of $\mathbf{C}$, one needs to have M separate $\mathbf{C}_m$ multiplications with $m \in [M]$.

Note that in [23], the authors consider weighted block sampling for column-times-row partitioning, and they optimize the sampling distribution based on the Frobenius norm of the multiplied sub-blocks. Similar to their study, one can decide the importance levels of the sub-blocks based on their actual Fronebius norms if they are available at the PS instead of using the block statistics to determine the importance levels.

B. UEP Coded Matrix Multiplication

For clarity, in the following we describe the UEP coding strategies based on $\mathsf{r}\times\mathsf{c}$ partitioning as given in (2). For

the case $\mathsf{c} \times \mathsf{r}$, the UEP coding is performed in an analogous manner and thus shall not be detailed further. We will consider the NOW-UEP and EW-UEP coding schemes in Sec. III-B, and apply them to the matrix multiplication problem in Sec. II.

The PS selects importance levels, and then encodes the corresponding rows and columns of $\mathbf{A}$ and $\mathbf{B}$ for the $\mathsf{r} \times \mathsf{c}$ multiplication using

$$\mathbf{W}_A^w = \sum_i \alpha^w(i)\mathbf{A}_{\pi_A^w(i)},$$
$$\mathbf{W}_B^w = \sum_j \beta^w(j)\mathbf{B}_{\pi_B^w(j)}, \tag{10}$$

for the w-th worker, $w \in [W]$, where $\alpha^w(i)$ and $\beta^w(j)$ are randomly selected elements from the given finite field, and $\pi_A^w(i)/\pi_B^w(j)$ is the row/column or column/row indices of $\mathbf{A}/\mathbf{B}$ at the corresponding levels, respectively.

After the matrix sub-products are encoded, $\mathbf{W}_A^w$ and $\mathbf{W}_B^w$ are transmitted to the w-th worker. Each worker w performs the corresponding multiplication operation resulting in $\mathbf{W}_A^w\mathbf{W}_B^w$, and transmits the sub-product to the PS. Since wireless transmission requires delay and energy, we consider the delay due to stragglers' channel conditions as the communication cost.

The PS will have $\mathcal{W}(T_{\max})$, which is a set of $\mathbf{W}_A^w\mathbf{W}_B^w$ products completed within a predetermined deadline $T_{\max}$. At this point, to form the approximation $\widehat{\mathbf{C}}$, the PS simply places the sub-product $\widehat{\mathbf{C}}_{nmp} = \mathbf{C}_{nmp}$ in the positions that can be obtained from $\mathcal{W}(T_{\max})$, and sets $\widehat{\mathbf{C}}_{nmp}$ to the all-zero matrix otherwise. With this operation, the final approximate matrix product $\widehat{\mathbf{C}}$ is obtained.

V. Analysis of the Approximation Error

In this section, we bound the loss in (1) using UEP codes. Our bounds rely on the results in [16] to characterize the performance of the proposed schemes in Sec. IV. These bounds are applied to the case of matrices with i.i.d. entries as per the following assumption.

Assumption 1 (Mutually Uncorrelated Matrix Entries): For the matrices $\mathbf{A}$ *and* $\mathbf{B}$*, we assume that the sub-matrices in the* $\mathsf{r} \times \mathsf{c}$ */* $\mathsf{c} \times \mathsf{r}$ *partitioning have uncorrelated entries with zero mean and variance* $\sigma_{l,A}^2$ *and* $\sigma_{l,B}^2$ *corresponding to the* l*-th importance level of the final product* $\mathbf{C}$*.*

A. Row-Times-Column Case

We assume for simplicity that the entries of the matrices are zero mean and with variance $\sigma_{A_n}^2$ and $\sigma_{B_p}^2$ for the n-th and p-th sub-block of (2), respectively, for $\mathbf{A}$ and $\mathbf{B}$, and they are uncorrelated, so that

$$\mathbb{E}\left[\|\mathbf{C}_{np}\|_F^2\right] = UHQ\sigma_{A_n}^2\sigma_{B_p}^2. \tag{11}$$

Let us denote the number of encoded matrix products received at time t by $N(t)$, then the probability of receiving w packets from W workers at time t is $P_{N(t)}(w)$, which is obtained as

$$P_{N(t)}(w) = \binom{W}{w}(1-F(t))^{W-w}\left(F(t)\right)^w. \tag{12}$$

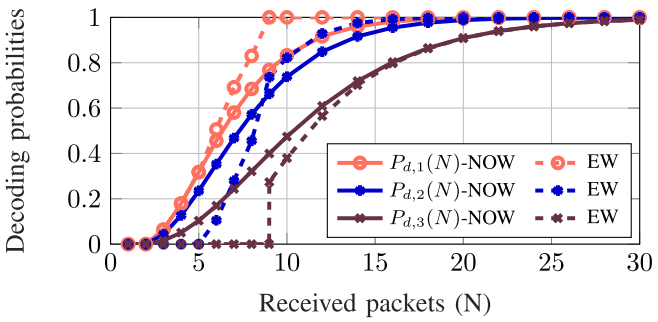


Fig. 7. Decoding probabilities of NOW-UEP and EW-UEP strategies with three classes, and $W = 30$ workers.

From [16, Eq. 5], we obtain a bound (which is achievable with large field sizes) on the decoding probabilities of the NOW-UEP strategy for each importance level as a function of the received matrices as

$$P_{d,l}(N) \leq \sum_{\substack{(n_1,n_2,\ldots,n_L)\\ \sum_l n_l = N}} P_{\Gamma(\xi),N}(\mathbf{n})\,\mathbf{1}(n_l \geq k_l), \tag{13}$$

where $\mathbf{n} = [n_1, n_2, \ldots, n_L]$, k_l is the number of packets in class l, and

$$P_{\Gamma(\xi),N}(\mathbf{n}) = \frac{N!}{n_1!n_2!\ldots n_L!}\Gamma_1^{n_1}\Gamma_2^{n_2}\ldots\Gamma_L^{n_L}. \tag{14}$$

It is also worth noting that the decoding probabilities for EW-UEP coding can be calculated similarly as they are given in [16, Eq. 6-9].

As an example, with three classes, $W = 30$ workers, and window selection probabilities $(0.40, 0.35, 0.25)$, the decoding probabilities of each class with NOW and EW-UEP codes with 3 sub-products in each level are as depicted in Fig. 7. The figure clearly illustrates how the most important class is protected better.

We can bound the performance of the coded matrix multiplication scheme in Sec. IV as follows.

Theorem 1 (UEP Loss for the Row-Times-Column Partitioning): Consider the loss minimization problem in Sec. II for the case in which the set of matrix products $\mathcal{D}(\{\mathbf{A},\mathbf{B}\})$ *that satisfy Assumption 1. The expected value of the loss in* (1) *attainable with the NOW-UEP strategy described in Sec. IV for* $\mathsf{r} \times \mathsf{c}$ *partitioning is*

$$\mathbb{E}\left[\mathcal{L}(T_{\max})\right] = \sum_w P_{N(T_{\max})}(w)\mathbb{E}[\|\mathbf{C}-\hat{\mathbf{C}}\|_F^2 | N(T_{\max}) = w], \tag{15}$$

with

$$\mathbb{E}[\|\mathbf{C}-\hat{\mathbf{C}}\|_F^2 | N(t) = w] = UHQ\sum_l k_l(1-P_{d,l}(w))\,\sigma_{l,A}^2\sigma_{l,B}^2, \tag{16}$$

where k_l *is the number of blocks in the* l*-th importance level of* $\mathbf{C}$*, and the expectation in* (15) *is taken over the random entries of* $\mathbf{A}$*,* $\mathbf{B}$*.*

Note that, since (13) is in fact an upper bound on the correct recovery probability, applying it to (15) results in a lower bound on the expected loss, however, this bound is tight as

the field size tends to infinity, i.e., the lower bound on the loss is asymptotically achievable. The analog of Theorem 1 for the EW-UEP can be obtained using corresponding decoding probabilities $P_{d,l}(N)$'s from the results in [16, Eq. 6-9]. They are not presented here for brevity.

Remark 4: Note that, in Theorem 1, there exists a "matching" between the probabilistic structure of the matrices to be multiplied and their $\mathsf{r} \times \mathsf{c}$ *block partitioning. In reality, one would not observe such a neat organization of the matrix values, and instead would have to fit the row/column weight distribution in the data to design the UEP code resulting in the minimal loss.*

B. Column-Times-Row Case

We now consider the case of column-times-row multiplication with partitioning (3). With a similar sub-block partitioning, we assume for simplicity that the entries of the matrix are zero mean and with variance $\sigma^2_{A_m}$ and $\sigma^2_{B_m}$ for the m-th sub-blocks of (3), respectively, for $\mathbf{A}$ and $\mathbf{B}$. Since $\mathbf{A}_m \in \mathbb{R}^{U\times H}$ and $\mathbf{B}_m \in \mathbb{R}^{H\times Q}$ for $m \in [M]$, the resulting multiplication is $\mathbf{C}_m \in \mathbb{R}^{U\times Q}$ with the same dimension of the original multiplication result $\mathbf{C}$. Note that, the Frobenius norm of $\mathbf{C}_m$ can be easily calculated as $\mathbb{E}\left[\|\mathbf{C}_m\|_F^2\right] = UHQ\sigma^2_{A_m}\sigma^2_{B_m}$. Similar to the previous section, the decoding probability (13) as given in [16, Eq. 5] is used for $\mathsf{c} \times \mathsf{r}$ matrix multiplication with NOW-UEP codes. Same approach can be extended to the EW-UEP code by using [16, Eq. 6-9]. Assuming that there are W workers where $N(t)$ encoded matrix products are received up to time t, we have the following analog of Theorem 1 which can be used with both NOW- and EW-UEP schemes by using the relevant decoding probabilities.

Theorem 2 (UEP Loss for the Column-Times-Row Multiplication): Consider the loss minimization problem in Sec. II for the case in which the set of matrix products $\mathcal{D}(\{\mathbf{A},\mathbf{B}\})$ *under Assumption 1. The expected value of the loss in* (1) *with the NOW-UEP strategy described in Sec. IV for* $\mathsf{c} \times \mathsf{r}$ *partitioning is*

$$\mathbb{E}\left[\mathcal{L}(T_{\max})\right] = \sum_w P_{N(T_{\max})}(w)\mathbb{E}[\|\mathbf{C}-\hat{\mathbf{C}}\|_F^2 | N(T_{\max}) = w], \tag{17}$$

with

$$\mathbb{E}[\|\mathbf{C}-\hat{\mathbf{C}}\|_F^2 | N(t) = w] \leq MUHQ\sum_l k_l(1-P_{d,l}(w))\sigma^2_{l,A}\sigma^2_{l,B}, \tag{18}$$

where k_l *is the number of blocks in the* l*-th importance level of* $\mathbf{C}$*, and the expectation in* (17) *is taken over the random entries of* $\mathbf{A}$*,* $\mathbf{B}$*.*

Proof: Consider the partitioning given in (3). Then,

$$\|\mathbf{C}-\hat{\mathbf{C}}\|_F \overset{(a)}{\leq} \sum_m \|\mathbf{C}_m - \hat{\mathbf{C}}_m\|_F, \tag{19a}$$

$$\|\mathbf{C}-\hat{\mathbf{C}}\|_F^2 \leq \left(\sum_m \|\mathbf{C}_m - \hat{\mathbf{C}}_m\|_F\right)^2, \tag{19b}$$

$$\overset{(b)}{\leq} M\sum_m \|\mathbf{C}_m - \hat{\mathbf{C}}_m\|_F^2, \tag{19c}$$

where (a) is the result of triangular inequality, and (b) follows from the Cauchy-Schwartz inequality. The proof is concluded by taking the expectation over the random entries by conditioning over the number of received packets at time t. □

TABLE III
UEP CODING PARAMETERS FOR THE MATRIX APPROXIMATION USED IN SECS. VI AND VII

	Class 1	Class 2	Class 3
# of blocks	3	3	3
Window selection probs.	0.40	0.35	0.25

VI. NUMERICAL EVALUATION WITH SYNTHETIC DATA

Let us begin by evaluating the multiplication in the $\mathsf{r}\times\mathsf{c}$ and $\mathsf{c}\times\mathsf{r}$ cases for a class of matrices satisfying Assumption 1. The task is performed with the help of $W = 30$ workers whose task completion times are modeled by exponential latency model with parameter $\lambda = 1$. We select $N = P = 3$, $U = Q = 300$, and $H = 900$ for the $\mathsf{r}\times\mathsf{c}$ case, and $U = Q = 900$, $H = 100$, and $M = 9$ for the $\mathsf{c}\times\mathsf{r}$ case for fairness of the comparisons of the two multiplication schemes in terms of the computational load of the workers.

For $\mathsf{r} \times \mathsf{c}$ multiplication, as discussed earlier, we classify each row and column blocks of $\mathbf{A}$ and $\mathbf{B}$ with importance levels *high*, *medium*, and *low*. The elements of each block are uncorrelated and distributed with $\mathcal{N}(0, 10)$, $\mathcal{N}(0, 1)$, and $\mathcal{N}(0, 0.1)$, for *high*, *medium*, and *low* levels, respectively. We assume that both $\mathbf{A}$ and $\mathbf{B}$ have only one instance of row and column from each level, i.e., $N = 3$, $P = 3$ with descending order of importance levels. $\mathbf{A}_1$ and $\mathbf{B}_1$ are from the *high* importance level; $\mathbf{A}_2$ and $\mathbf{B}_2$ are from the *medium* importance level; $\mathbf{A}_3$ and $\mathbf{B}_3$ are from the *low* importance level. We take the multiplication of (*high* and *high*) and (*high* and *medium*) blocks as *class one*, (*medium* and *medium*) and (*high* and *low*) blocks as *class two*, and the remaining as *class three*. With this definition, we have $(k_1, k_2, k_3) = (3, 3, 3)$ sub-blocks in each class.

For $\mathsf{c} \times \mathsf{r}$ multiplication, we assume that $\mathbf{A}_i$ and $\mathbf{B}_i$ are from the *high* importance level if $i \in \{1, 2, 3\}$, from *medium* importance level if $i \in \{4, 5, 6\}$, and from *low* importance level if $i \in \{7, 8, 9\}$. The multiplication of (*high* and *high*) blocks are considered as *class one*, (*medium* and *medium*) blocks as *class two* and (*low* and *low*) blocks as *class three*, resulting in three sub-blocks in each class.

In our simulations, we select the window selection probabilities for both NOW and EW-UEP strategies as $(\Gamma_1, \Gamma_2, \Gamma_3) = (0.40, 0.35, 0.25)$ for both $\mathsf{r} \times \mathsf{c}$ and $\mathsf{c} \times \mathsf{r}$ cases as shown in Table III. The decoding probabilities for each class are obtained through the formulation given in [16], as also depicted for the NOW-UEP and EW-UEP strategies in Fig. 7. As expected, the first class has higher decoding probability with both NOW and EW-UEP strategies.

In Fig. 8, these decoding probabilities are used to obtain the normalized expected loss values of $\mathsf{r} \times \mathsf{c}$ and $\mathsf{c} \times \mathsf{r}$ as a function of time t along with the performance obtained with the Maximum Distance Separable (MDS) codes which are also used in [9] for coded computation. By utilizing MDS

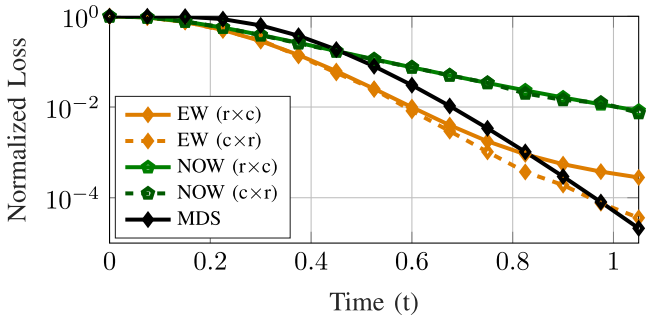


Fig. 8. Normalized loss of the estimator using UEP codes with three classes with exponential latency model.

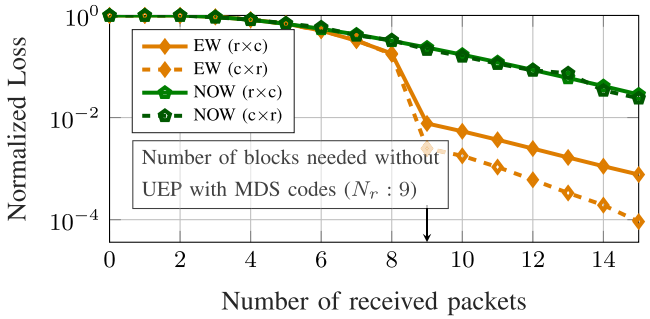


Fig. 9. Normalized loss of the estimator using UEP codes with three classes as a function of number of received packets.

codes, the perfect reconstruction is achieved when the number of received packets is equal to the total number of sub-blocks. Until time $t = 0.44$, the UEP protection with NOW scheme performs better than that of MDS with both $\mathsf{r}\times\mathsf{c}$ and $\mathsf{c}\times\mathsf{r}$, since it enables an early recovery of important classes with a small number of received packets. With the EW-UEP protection, we have higher protection for more important classes which gives a better approximation than the MDS coding until time $t = 0.825$ and $t = 0.975$, respectively, with $\mathsf{r} \times \mathsf{c}$ and $\mathsf{c} \times \mathsf{r}$ multiplication schemes. This also shows that $\mathsf{c} \times \mathsf{r}$ multiplication scheme is more efficient than $\mathsf{r}\times\mathsf{c}$ multiplication scheme. In other words, if we are interested in an earlier recovery of certain important parts, using the UEP coding approach for matrix approximation is highly advantageous, especially with EW-UEP and $\mathsf{c} \times \mathsf{r}$ multiplication. After time $t = 0.975$, the MDS code starts to perform better than all the others since it can fully recover $\mathbf{C}$ after receiving nine packets. If we wait long enough, the UEP strategy will also fully recover the desired matrix product.

For further interpretation, we give the normalized loss values of matrix multiplication with MDS coding and approximate matrix multiplication using NOW and EW-UEP coding with both $\mathsf{r} \times \mathsf{c}$ and $\mathsf{c} \times \mathsf{r}$ in Fig. 9 as a function of the number of received packets. The matrix multiplication with MDS codes needs to receive $\sum_l k_l$ packets to fully recover the result, where k_l is the number of packets in the l-th level. Receiving less than $\sum_l k_l$ will not provide any partial information, and results in no recovery, hence the normalized loss with MDS coding is unity until it receives nine packets (the minimum required for recovery). However, matrix product

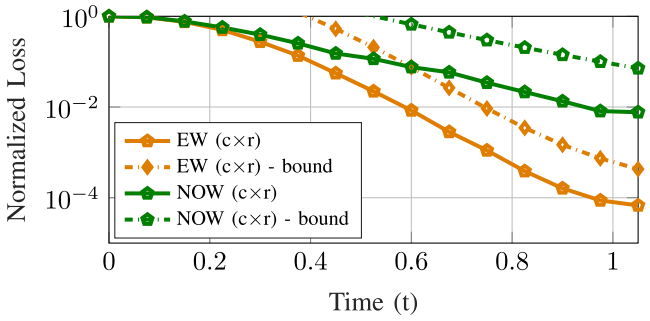


Fig. 10. Normalized loss of the estimator using UEP codes with $\mathsf{c}\times\mathsf{r}$ multiplication and its upper bound.

approximation with NOW and EW-UEP coding strategies start to recover more important classes after receiving only very few packets, and continue to provide additional partial information after each received block. It is also important to note that the loss formulation given in Theorem 1 for $\mathsf{r}\times\mathsf{c}$ is an exact result that is achievable with large field sizes. Hence the simulation results given in both Figs. 8 and 9 for $\mathsf{r} \times \mathsf{c}$ case match our theoretical expectations.

Furthermore, in Fig. 10, we compare the upper bound of the loss of $\mathsf{c} \times \mathsf{r}$ multiplication given in Theorem 2 with simulation results using NOW and EW-UEP codes. As shown in the figure, the upper bound is not tight, however, it can be used to illustrate the behavior of the loss curve, that is, they successfully reflect the loss behavior of the proposed scheme. Thus, one can use these bounds to examine the difference between NOW-UEP and EW-UEP schemes or their performance with different window selection probabilities. It is also worth noting that we have chosen the window selection distributions for the UEP codes arbitrarily, and they can be further optimized for improved performance.

VII. DNN Back-Propagation With Approximate Matrix Multiplication

In this section, we consider an application of the approximate matrix multiplication approach in Sec. IV to the training of DNNs on two classification datasets: the MNIST handwritten digits [34] and the CIFAR-10 images [35]. The next sub-section introduces the DNN settings, Sec. VII-B numerically validates the sparsity in gradient matrices, and Sec. VII-C presents the relevant training results.

A. DNN Settings

In this section, we consider the training for the MNIST and CIFAR-10 datasets. The MNIST dataset requires a quite simple DNN structure with three dense layers followed by a softmax layer, and training requires only a few epochs to attain a high classification accuracy. Unlike the MNIST, the CIFAR-10 dataset has more features, and thus, it requires both a more complex DNN architecture and more epochs for learning. In this DNN model, we use two 2D convolution layers followed by a 2D max-pooling. After the max-pooling, the processed images pass through the fully connected layers, where we employ three dense layers.

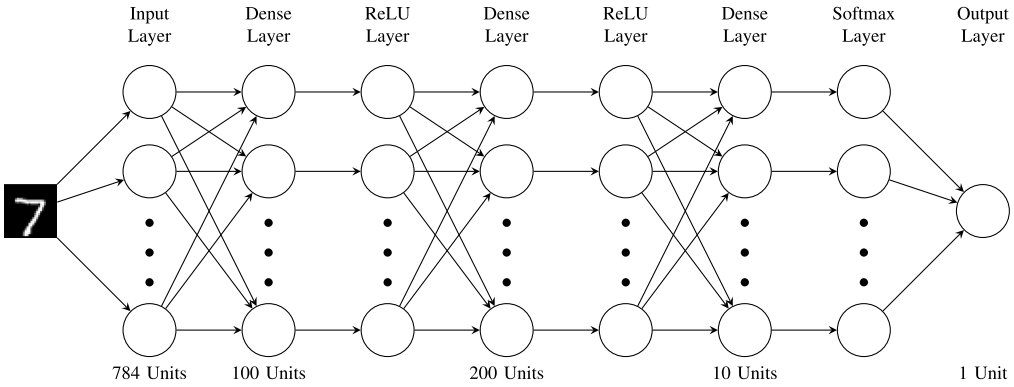


Fig. 11. DNN used for classifying the MNIST dataset in Sec. VII.

TABLE IV
PARAMETERS AND HYPERPARAMETERS USED FOR THE TRAINING OF THE DNN MODELS

	MNIST	CIFAR-10
Training Samples	60k	50k
Optimizer	SGD	
Learning Rate	0.01	
Loss	Categorical Cross Entropy	
Epochs	3	120
Mini-Batch Sizes	64	

TABLE V
A SUMMARY OF THE MODEL LAYERS OF THE DNN USED FOR CIFAR-10 CLASSIFICATION

Layer Name	Kernel/Weights	Padding	Activation Layer
Conv2D 1	$3 \times 3 \times 32$	Same	ReLU
Conv2D 2	$3 \times 3 \times 32$	Valid	ReLU
MaxPooling2D	2×2	-	-
Flatten	-	-	-
Dense 1	7200×512	-	ReLU
Dense 2	512×256	-	ReLU
Dense 3	256×12	-	Softmax

A representation of the network structure for the MNIST digit classification task is provided in Fig. 11. The overall network structure for the CIFAR-10 classification task is defined in Table V. Further parameters utilized in the numerical evaluations are provided in Table IV.

Let us now denote the network as D and the total number of layer as I. Let us further denote the weights, bias, inputs, outputs, and gradients of the corresponding layer as $\mathbf{V}_i$, $\mathbf{b}_i$, $\mathbf{X}_i$, $\mathbf{O}_i$, and $\mathbf{G}_i$. From this we can define the output of each layer as $\mathbf{O}_i = D_i(\mathbf{X}_i)$, and the output of the network as $\mathbf{O} = D(\mathbf{X})$, where $\mathbf{X} = \mathbf{X}_1$ is the initial input and $\mathbf{O} = \mathbf{O}_I$ is the final output. Note that $\mathbf{O}_i = \mathbf{X}_{i+1}$. Then the forward propagation of a dense layer can be defined as

$$D_i(\mathbf{X}_i) = \mathbf{X}_i\mathbf{V}_i + \mathbf{b}_i. \tag{20}$$

However, for the back-propagation we must consider two matrix multiplications

$$\mathbf{G}_i = \mathbf{G}_{i+1}\mathbf{V}_i^T, \tag{21}$$

TABLE VI
A SUMMARY OF THE BACK-PROPAGATION MATRICES FOR THE DNN MODELS. FOR $+$, SEE REMARK 5

MNIST		
Layer	$\mathbf{G}_i$	$\mathbf{V}_i^*$
Dense 1	$(64 \times 100) \cdot (100 \times 784)$	$(784 \times 64) \cdot (64 \times 100)$
Dense 2	$(64 \times 200) \cdot (200 \times 100)$	$(100 \times 64) \cdot (64 \times 200)$
Dense 3	$(64 \times 10) \cdot (10 \times 200)$	$(200 \times 64) \cdot (64 \times 10)$
CIFAR-10		
Layer	$\mathbf{G}_i$	$\mathbf{V}_i^*$
Conv 1/2	+	+
Dense 1	$(64 \times 512) \cdot (512 \times 7200)$	$(7200 \times 64) \cdot (64 \times 512)$
Dense 2	$(64 \times 256) \cdot (256 \times 512)$	$(512 \times 64) \cdot (64 \times 256)$
Dense 3	$(64 \times 10) \cdot (10 \times 256)$	$(256 \times 64) \cdot (64 \times 10)$

and

$$\mathbf{V}_i^* = \mathbf{X}_i^T\mathbf{G}_{i+1}, \tag{22}$$

where $\mathbf{V}_i^*$ is used for updating $\mathbf{V}_i$ and (21) for calculating the $\mathbf{G}_i$ used in D_{i-1} for (22). $\mathbf{G}_L$ is calculated from the derivative of the loss between $\mathbf{O}_L$ and $\mathbf{Y}$, the ground truth.

The dimensions of these matrices are defined in Table VI for both the MNIST and CIFAR-10 DNN models. The reason we focus on these matrices will be explained in Sec VII-B.

Remark 5: In our simulations we apply the proposed approximate matrix multiplication method only to the dense layer's back-propagation. An extension of the proposed method to the convolutional layers is left for future research.

B. Sparsity of DNN Gradients

Generally speaking, as the training of a DNN proceeds toward convergence, weight updates become increasingly smaller and thus gradients converge to zero. In order to introduce resilience against numerical errors and encourage sparsity in the trained model, gradient sparsification is often applied. Broadly speaking, gradient sparsification is defined as the setting of some elements of the gradient updates to zero according to some policy. For simplicity, let us assume that sparsification takes the form of a simple thresholding, that is,

$$R(x) = \begin{cases} x, & |x| > \tau \\ 0, & |x| \leq \tau, \end{cases} \tag{23}$$

where $\tau \in [Z]$ is the threshold and x is an element of matrix $\mathbf{A/B}$. The value of τ is chosen close to the machine precision at the beginning of training and is increased at each epoch. Also, τ is chosen differently for each layer: making the shallow layers less sparse than the deeper ones. As mentioned in [7], DNNs are fault-tolerant and resilient to such approximations in the gradient evaluations.

In applying the method in Sec. IV, we shall rely on the gradient matrix sparsity to define the protection level of each packet. Consider the DNN for the MNIST classification task described in Sec. VII-A, and use $\tau = 10^{-5}$ for sparsification of the gradient inputs, and $\tau = 10^{-4}$ for the weights and inputs of each layer. This choice of τ at mini-batch iteration 389 / 937 of the first epoch and for each layer results in the sparsity levels as given in Table II. The empirical distribution of the remaining weights very much resembles that of a Gaussian distribution with zero mean and a variance which increases with the depth of the layer. A Gaussian fitting of these non-sparse values are presented in Fig. 4. It can be observed that, indeed, the sparsity increases with the layer depth, as well as the variance of the fitted Gaussian. This shows that exploiting the variability in the matrix sub-block norms to apply unequal error protection has the potential of drastically improving the training performance.

C. DNN Performance With UEP Coded Matrix Multiplication

In this section, we investigate the accuracy of the proposed solution for the two DNN models described in Sec. VII-A. In our comparisons, we show the performance of the DNN training with

- **red line:** centralized computation with no stragglers,
- **green line:** distributed computation with NOW-UEP codes,
- **yellow line:** distributed computation with EW-UEP codes,
- **purple line:** distributed computation with 2-block repetitions.

In all the above cases,

- **continuous line:** is for the $\mathsf{r} \times \mathsf{c}$ paradigm, while
- **dotted line:** is for the $\mathsf{c} \times \mathsf{r}$ paradigm.

For a fair comparison among these scenarios, we scale the time required to complete a task as $F(\Omega t)$, where Ω is the number of matrix sub-products divided by the number of workers (*see Remark 1*). For the simulations, we have that the total number of matrix sub-products is 9, where $N = P = 3$, thus $NP = 9$ for $\mathsf{r} \times \mathsf{c}$ and $M = 9$ for $\mathsf{c} \times \mathsf{r}$. Additionally, we consider an exponential latency model with $\lambda = 0.5$. We take $T_{\max} \in \{0.25, 0.5, 1, 2\}$. Other simulation parameters are given in Tables VII and VIII. The dimension of the encoding matrices $\mathbf{A/B}$ for each dense layer and gradient are shown in Table VI. These settings are for both the $\mathsf{r} \times \mathsf{c}$ and the $\mathsf{c} \times \mathsf{r}$ cases.

Next, let us describe how the importance levels are obtained in our coding scheme. Similar to [36], which proposes a fast matrix multiplication algorithm, the column/row indexes are permuted so as to obtain a descending magnitude of the column/row weights. Note that ordering has average complexity $\mathcal{O}(n \log n)$ in the number of columns/rows, so that the computational burden at the PS is minimal as compared to the matrix multiplication complexity. Once ordered in decreasing magnitudes, column/row sub-blocks are formed by dividing them into three groups of (roughly) equal size. The sub-blocks are then encoded using the NOW/EW-UEP code as specified in Table III.

TABLE VII
A SUMMARY OF THE ENCODING PARAMETERS IN SEC. VII-C

Encoding Type	W	Ω
Uncoded	9	9/9
NOW/EW - UEP	15	9/15
2-Block Rep	18	9/18

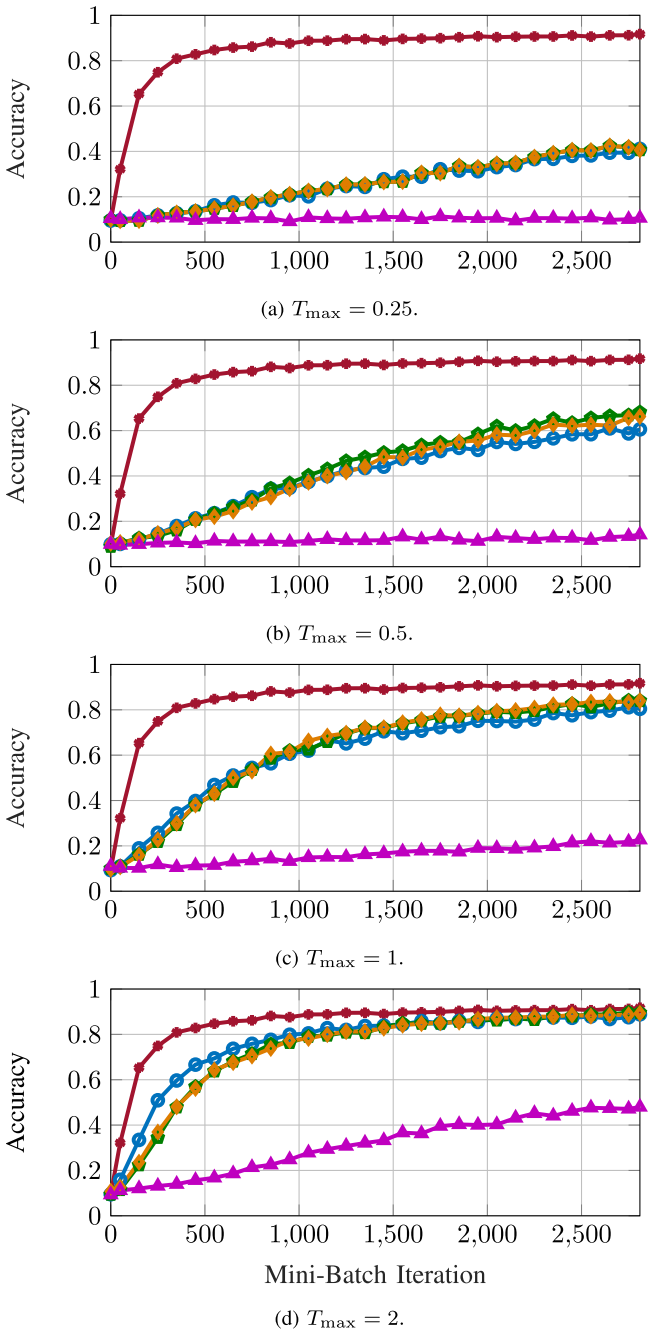
TABLE VIII
A SUMMARY OF THE NUMBER OF SUB-BLOCKS BELONGING TO EACH IMPORTANCE LEVEL FOR UEP CODES USED IN SEC. VII-C

Importance Level	$n_C(s)$
High	1
Medium	2
Low	6

1) MNIST [34]: Figs. 12 and 13 depict the results of using different multiplication strategies for different $T_{\max}$ values, $\mathsf{r} \times \mathsf{c}$ for the former and $\mathsf{c} \times \mathsf{r}$ for the latter with the MNIST dataset, and Fig. 14 serves as a merger of these two by fixing the mini-batch iteration to explore the accuracy trade-off when using different deadlines. Ideally, we would want to increase the accuracy while decreasing the deadline so that the model can achieve convergence in the least time possible. Note that since the case with no-stragglers receives all the sub-blocks back by any deadline, its accuracy is a constant and it is used as our benchmark.

We observe from Fig. 14 that for $T_{\max} < 2$, the UEP coding strategy shows an advantage over the others, with a more significant accuracy gap with $\mathsf{c} \times \mathsf{r}$ at $T_{\max} = 0.5$ and 1. By selecting two deadlines that are multiples of each other by a factor of x in Figs. 14a and 15b, such as 1 and 2 by a factor of 2, we can observe a clear trade-off in the accuracy: if it takes n iterations for the longer deadline to achieve a y accuracy, it will take x times n iterations for the shorter deadline to achieve an accuracy close to y. However, further study is necessary to determine whether this holds true with other datasets and DNN architectures.

Although the uncoded scheme provides no protection against stragglers, missing a few of blocks does not cause much damage to the learning process, showing once again the inherent fault-tolerance of DNNs. Perhaps not too surprisingly, employing block repetition coding increases the number of workers required but does not result in a better overall performance as compared to the uncoded scheme. This result follows from the choice of the waiting time distribution and it provides an understanding on the computational complexity scaling in our simulations, as discussed in Remark 1. Consider the following two scenarios: (i) one machine completes a job, versus (ii) the same job is given to two machines and the job is completed whenever one of the two machines finishes.



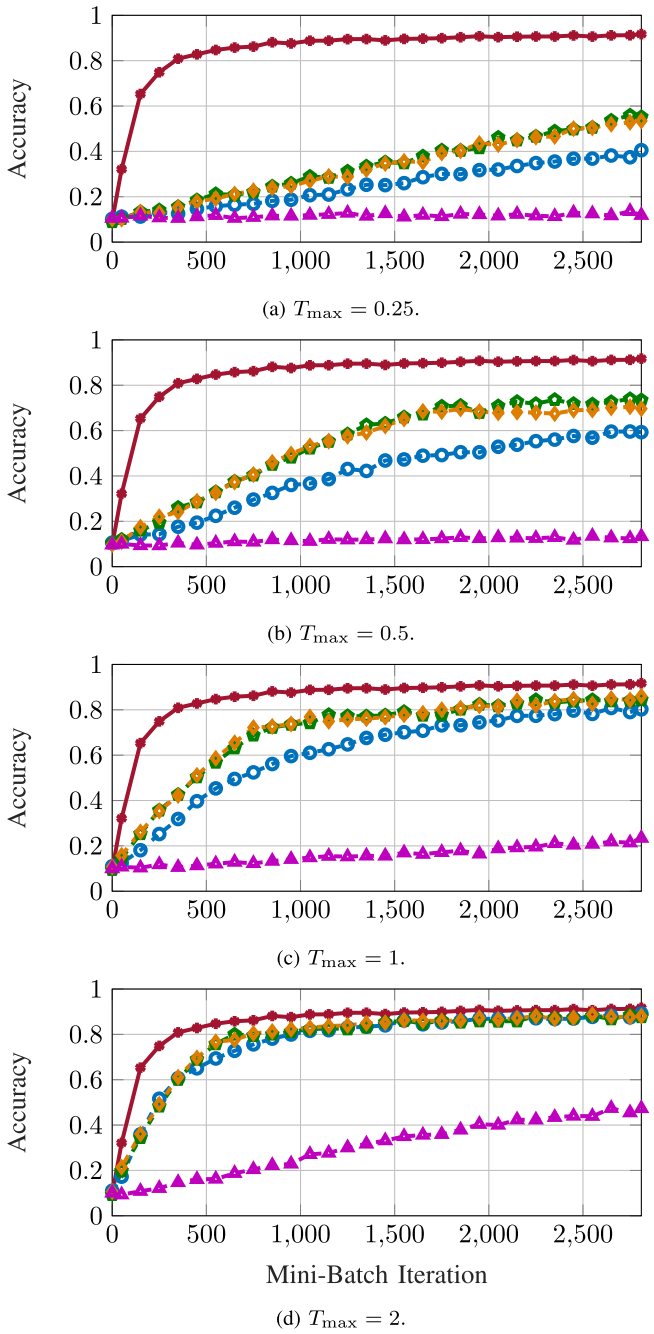
(a) $T_{\max} = 0.25$.

(b) $T_{\max} = 0.5$.

(c) $T_{\max} = 1$.

(d) $T_{\max} = 2$.

Fig. 12. MNIST classification accuracy using row-times-column matrix multiplication for the DNN proposed in Sec. VII-A.



(a) $T_{\max} = 0.25$.

(b) $T_{\max} = 0.5$.

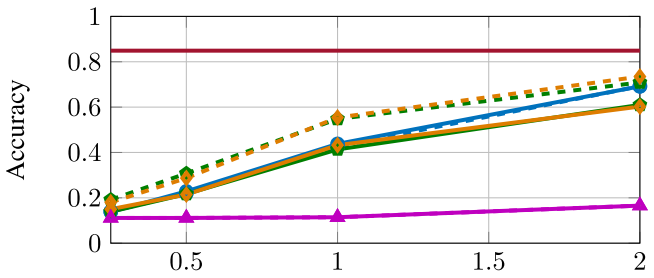
(c) $T_{\max} = 1$.

(d) $T_{\max} = 2$.

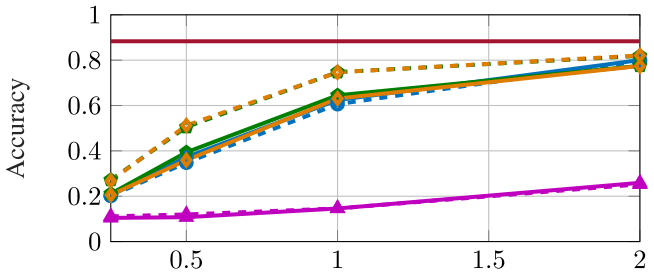
Fig. 13. MNIST classification accuracy using column-times-row matrix multiplication for the DNN proposed in Sec. VII-A.

To compare these two scenarios in a fair manner, the expected value of the waiting time of the second scenario must be twice of the first one. For our choice of waiting time distribution, the scenario (i) performs better than the scenario (ii) above, that is, there is no intrinsic advantage in distributing a given job to multiple workers. From Fig. 14, we observe that $\mathsf{r} \times \mathsf{c}$ and $\mathsf{c} \times \mathsf{r}$ partitioning have roughly the same performance as uncoded and 2-block repetition. However, in the case of UEP codes, $\mathsf{c} \times \mathsf{r}$ shows an advantage over $\mathsf{r} \times \mathsf{c}$, i.e., mini-batch 1000 and 1500 for $T_{\max} = 0.25$ to 1. Further exploration of this phenomenon is needed to understand whether this is true for this specific DNN or it is more general.

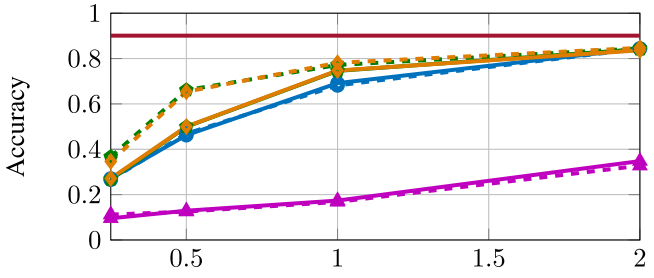
2) CIFAR-10 [35]: For the simulations with the CIFAR-10 dataset in Fig. 1, we use $\lambda = 0.5$ with an exponential latency model and $T_{\max} = 1$. Since the model evolves slowly, the first few epochs result in gradient computations that are rather uniform in row/column norms. For this reason, we let the model train for the first 30 epochs without stragglers. After these first set of epochs, there is enough sparsification in the gradients to justify UEP coding, and accordingly, Fig. 1 presents the simulation results between 30 and 120 epochs. The calculations for the convolutional layer are performed without stragglers through central computations. For the dense layers, we use the coding strategy summarized in Table VII



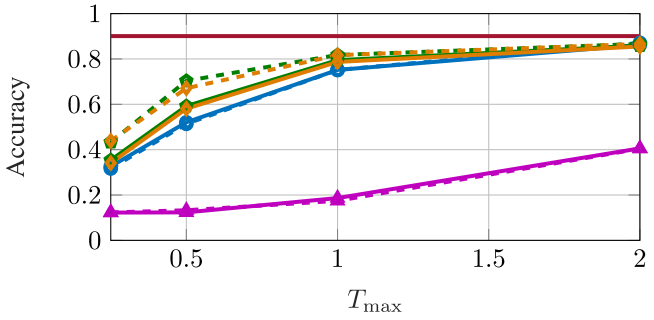
(a) Mini-batch iteration = 500.



(b) Mini-batch iteration = 1000.



(c) Mini-batch iteration = 1500.



(d) Mini-batch iteration = 2000.

Fig. 14. MNIST classification accuracy at different $T_{\max}$ for the DNN proposed in Sec. VII-A.

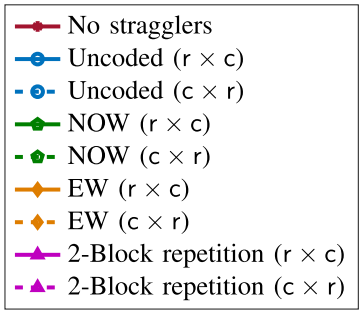


Fig. 15. Legend for Fig. 12 to Fig. 14.

and the corresponding scaling of the waiting time distribution. The only exception is the set of matrices in (22) for the last layer, for which we use the uncoded scheme since they were not sufficiently sparse.

As we can observe from Fig. 1, after around 60 epochs, the UEP codes widen the accuracy margin and learn faster than the other two encoding strategies. This is in part due to the gradient evolution where the weights of the sub-blocks show a higher variance than they did between epochs 30 to 60 due to sparsification and convergence. This effect continues to amplify as the epoch progresses, promoting the need of unequal error protection. In particular, it appears that both classes of the UEP codes employed yield an accuracy which tends to those attainable without stragglers, i.e., 1, while the uncoded transmission performance saturates below 0.9. It is interesting to note that this situation is substantially different from the scenario in Fig. 12 in which the MNIST dataset DNN is trained with the same value of $T_{\max}$. In the latter case, in fact, no substantial improvement is provided by the UEP codes over the uncoded transmission in terms of eventual accuracy.

We close this section by noting that all the simulation results can be reproduced using Matlab and Python codes available at: https://github.com/HernandezEduin/UEP-Straggler-Mitigation.

D. Future Work

In Sec. VII we apply the distributed approximate matrix computation scheme in Sec. IV to the evaluation of gradients through back-propagation for training a DNN. The performance of the proposed approach is evaluated only through numerical simulations, and a more detailed analysis of the relationship between distributed computation and distributed learning is left for future research. Here we would like to point out some aspects of this problem which we believe deserve further investigation.

1) Error Feedback: in the current implementation for DNN training, the back-propagation algorithm does not account for the specific nature of the gradient noise arising from the application of the scheme in Sec. IV. A more efficient solution can be implemented by accounting for the accumulation of the approximation error at each gradient evaluation similar to the approach in [37].

2) Computation Accuracy/Learning Accuracy: DNN training requires a successive evaluation of matrix products. Currently, it is not clear how the error in approximate computation propagates through these successive evaluations. Additionally, it is not well-understood how the error in the gradient approximation affects the learning accuracy at different iterations. Once these aspects have been better understood, the scheme in Sec. VII can be optimally designed at each DNN layer and at each iteration to yield the best expected accuracy by a chosen deadline.

3) Computation Time: The current modeling of the computation delay in (7) is an accurate model for queuing delays and might not properly account for the computation time requirements more specific to DNN training. By more appropriately

choosing the response times, one could obtain a more practical design for the scheme in Sec. IV.

4) Computation Load: Inherent in the design of the scheme in Sec. IV is the assumption that the computational cost of the matrix summation and scaling is negligible when compared to that of matrix multiplication. Accordingly, the PS can perform any type of coding which only employs linear combinations of matrices. In actuality, the computational load might be better expressed in terms of memory allocation or overall number of flops. In this scenario, the design of the scheme in Sec. IV should be modified to account for such limitations.

5) Extension to Federated Learning (FL): The current coding scheme is implemented in a non-standard distributed learning fashion, where computations are distributed from the PS to the workers on a per layer basis of the DNN. One can imagine the extension of the scheme to the FL setting in which the computational nodes (workers) have access to the training dataset through the cloud or have them locally available. In such a scenario, the PS would have its complexity reduced even further with the use of UEP codes.

6) Optimization of UEP Codes: In our work, we did not perform any optimization of the UEP codes, instead we chose the window selection distributions for both the NOW-UEP and EW-UEP schemes arbitrarily. An interesting direction for future research is the UEP code optimization for matrix product approximation. To do this, the window selection probabilities can be optimized to minimize the loss in the matrix approximation for the given setup. For instance, in the distributed DNN model, one should optimize these probabilities to minimize the error according to the current sub-weights.

VIII. Conclusion

In this paper, we have studied distributed approximate matrix multiplication using UEP codes with the objective of mitigating the straggler phenomenon. The proposed approach has a wide range of applications as it allows one to speed up large-scale operations which are common in machine learning and data mining algorithms. We use UEP codes to provide better protection for the sub-operations which have higher effects on the resulting matrix product. We validate the effectiveness of the proposed approach through analytical assessments based on simplified models for sparse matrices, and compare our results with those obtained with MDS codes via simulations. Furthermore, the proposed strategy is applied to the back-propagation steps of a DNN for the MNIST digit classification and CIFAR-10 image classification tasks. Our results clearly show that, in the presence of stragglers, we can have a performance close to the centralized training earlier by striking a balance between the precision of the updates and the response time of the edge devices.

References

[1] B. Tegin, E. E. Hernandez, S. Rini, and T. M. Duman, "Straggler mitigation through unequal error protection for distributed matrix multiplication," in *Proc. IEEE Int. Conf. Commun.*, Montreal, QC, Canada, Jun. 2021, pp. 1–6.

[2] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Trans. Inf. Theory*, vol. 65, no. 7, pp. 4227–4242, Jul. 2019.

[3] J. Choi, D. W. Walker, and J. J. Dongarra, "Pumma: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers," *Concurrency, Pract. Exper.*, vol. 6, no. 7, pp. 543–570, Oct. 1994.

[4] R. A. Van De Geijn and J. Watts, "SUMMA: Scalable universal matrix multiplication algorithm," *Concurrency, Pract. Exper.*, vol. 9, no. 4, pp. 255–274, Apr. 1997.

[5] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "OverSketch: Approximate matrix multiplication for the cloud," in *Proc. IEEE Int. Conf. Big Data*, Seattle, WA, USA, Dec. 2018, pp. 298–304.

[6] J. Kim, M. Son, and K. Lee, "MPEC: Distributed matrix multiplication performance modeling on a scale-out cloud environment for data mining jobs," *IEEE Trans. Cloud Comput.*, early access, Oct. 30, 2019, doi: 10.1109/TCC.2019.2950400.

[7] B. Plancher, C. D. Brumar, I. Brumar, L. Pentecost, S. Rama, and D. Brooks, "Application of approximate matrix multiplication to neural networks and distributed SLAM," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Waltham, MA, USA, Sep. 2019, pp. 1–7.

[8] M. Son and K. Lee, "Distributed matrix multiplication performance estimator for machine learning jobs in cloud computing," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, San Francisco, CA, USA, Jul. 2018, pp. 638–645.

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

[10] D. Wang, G. Joshi, and G. Wornell, "Using straggler replication to reduce latency in large-scale parallel computing," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 3, pp. 7–11, Nov. 2015.

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

[12] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with D-dimensional product codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Vail, CO, USA, Jun. 2018, pp. 1993–1997.

[13] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: An optimal design for high-dimensional coded matrix multiplication," in *Proc. 31th Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, Dec. 2017, pp. 4406–4416.

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

[15] A. Kosta, N. Pappas, and V. Angelakis, "Age of information: A new concept, metric, and tool," *Found. Trends Netw.*, vol. 12, no. 3, pp. 162–259, 2017.

[16] D. Vukobratović and V. Stanković, "Unequal error protection random linear coding strategies for erasure channels," *IEEE Trans. Commun.*, vol. 60, no. 5, pp. 1243–1252, May 2012.

[17] 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. 1–23, Jun. 2019.

[18] S. K. Hanna, R. Bitar, P. Parag, V. Dasari, and S. El Rouayheb, "Adaptive distributed stochastic gradient descent for minimizing delay in the presence of stragglers," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Barcelona, Spain, May 2020, pp. 4262–4266.

[19] R. Bitar, P. Parag, and S. E. Rouayheb, "Minimizing latency for secure distributed computing," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 2900–2904.

[20] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Aachen, Germany, Jun. 2017, pp. 2418–2422.

[21] A. Frieze, R. Kannan, and S. Vempala, "Fast Monte-Carlo algorithms for finding low-rank approximations," *J. ACM*, vol. 51, no. 6, pp. 1025–1041, Nov. 2004.

[22] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication," *SIAM J. Comput.*, vol. 36, no. 1, pp. 132–157, Jan. 2006.

[23] N. Charalambides, M. Pilanci, and A. O. Hero, "Approximate weighted CR coded matrix multiplication," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Toronto, ON, Canada, Jun. 2021, pp. 5095–5099.

[24] S. Dasgupta and A. Gupta, "An elementary proof of a theorem of Johnson and Lindenstrauss," *Random Struct. Algorithms*, vol. 22, no. 1, pp. 60–65, 2003.

[25] T. Sarlos, "Improved approximation algorithms for large matrices via random projections," in *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, Pittsburg, PA, USA, Oct. 2006, pp. 143–152.
[26] K. L. Clarkson and D. P. Woodruff, "Numerical linear algebra in the streaming model," in *Proc. 41st Annu. ACM Symp. Symp. theory Comput.*, Bethesda, MD, USA, 2009, pp. 205–214.
[27] D. M. Kane and J. Nelson, "Sparser Johnson-Lindenstrauss transforms," *J. ACM*, vol. 61, no. 1, pp. 1–23, Jan. 2014.
[28] K. L. Clarkson and D. P. Woodruff, "Low-rank approximation and regression in input sparsity time," *J. ACM*, vol. 63, no. 6, pp. 1–45, Feb. 2017.
[29] B. Masnick and J. Wolf, "On linear unequal error protection codes," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 4, pp. 600–607, Oct. 1967.
[30] N. Thomos, J. Chakareski, and P. Frossard, "Randomized network coding for uep video delivery in overlay networks," in *Proc. IEEE Int. Conf. Multimedia Expo*, New York, NY, USA, Jun. 2009, pp. 730–733.
[31] K. Nguyen, T. Nguyen, and S.-C. Cheung, "Video streaming with network coding," *J. Signal Process. Syst.*, vol. 59, no. 3, pp. 319–333, Jun. 2010.
[32] H. Shojania and B. Li, "Random network coding on the iPhone: Fact or fiction?" in *Proc. 18th Int. Workshop Netw. Oper. Syst. Support Digit. Audio Video*, Williamsburg, VA, USA, 2009, pp. 37–42.
[33] P. Vingelmann, F. H. P. Fitzek, M. V. Pedersen, J. Heide, and H. Charaf, "Synchronized multimedia streaming on the iphone platform with network coding," in *Proc. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, Jan. 2011, pp. 875–879.
[34] Y. LeCun, C. Cortes, and C. Burges, *MNIST Handwritten Digit Database*, vol. 2. Atlanta, GA, USA: ATT Labs, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist
[35] A. Krizhevsky, V. Nair, and G. Hinton. (2009). *CIFAR-10 (Canadian Institute for Advanced Research)*. Accessed: Feb. 15, 2021. [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html
[36] R. Yuster and U. Zwick, "Fast sparse matrix multiplication," *ACM Trans. Algorithms*, vol. 1, no. 1, pp. 2–13, 2005.
[37] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," 2018, *arXiv:1809.07599*. [Online]. Available: http://arxiv.org/abs/1809.07599

Busra Tegin (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical and electronics engineering from Bilkent University, Ankara, Turkey, in 2017 and 2020, respectively, where she is currently pursuing the Ph.D. degree with the Department of Electrical and Electronics Engineering. Her research interests include the general area of wireless communications, with an emphasis in federated learning and distributed computing.

Eduin E. Hernandez received the B.S. degree from the Electrical and Computer Engineering Department, National Chiao Tung University (NCTU), Taiwan, in 2019. He is currently pursuing the M.S. degree with the Electrical Engineering and Computer Science International Graduate Program, National Yang Ming Chiao Tung University (NYCU), Taiwan. His research interests include machine learning, distributed learning, approximation algorithms, and coded computing.

Stefano Rini received the B.A. degree in computer science from the Politecnico di Milano, Italy, in 2005, and the M.S. degree in electrical and computer engineering and statistics and the Ph.D. degree in electrical and computer engineering from the University of Illinois at Chicago (UIC), USA, in 2009 and 2011, respectively. In 2012, he was a Post-Doctoral Fellow at the Institute for Communications Engineering, Technical University of Munich (TUM), Germany, with Prof. Kramer. In 2013, he was a Post-Doctoral Fellow at the Department of Electrical Engineering, Stanford University, USA, with Prof. Goldsmith. In 2014, he joined the National Chiao Tung University (NCTU), Taiwan. He is currently an Associate Professor with the Department of Electrical and Computer Engineering (ECE), National Yang Ming Chiao Tung University (NYCU), formerly known as NCTU. His current research interests are in systems, with particular focus on communication and signal processing, including wireless and mobile communications, coding/modulation, coding for wireless communications, and spectroscopy.

Tolga M. Duman (Fellow, IEEE) received the B.S. degree in electrical engineering from Bilkent University, Ankara, Turkey, in 1993, and the M.S. and Ph.D. degrees in electrical engineering from Northeastern University, Boston, MA, USA, in 1995 and 1998, respectively. Prior to joining Bilkent University in September 2012, he was a Professor with the School of ECEE, Arizona State University. He is currently a Professor with the Electrical and Electronics Engineering Department, Bilkent University. His current research interests are in systems, with particular focus on communication and signal processing, including wireless and mobile communications, coding/modulation, coding for wireless communications, and machine learning. He was a recipient of the National Science Foundation CAREER Award and the IEEE Third Millennium Medal. He is the Editor-in-Chief of IEEE TRANSACTIONS ON COMMUNICATIONS.