2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP) | 979-8-3503-2411-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/MLSP55844.2023.10285915

CODED DISTRIBUTED IMAGE CLASSIFICATION

Jiepeng Tang[†], *Navneet Agrawal*[‡], *Slawomir Stanczak*[‡⋆], *Jingge Zhu*[†]

[†] University of Melbourne, Australia, [‡] Technische Universität Berlin, Germany,
[⋆] Fraunhofer Heinrich Hertz Institute, Germany

ABSTRACT

In this paper, we present a coded computation (CC) scheme for distributed computation of the inference phase of machine learning (ML) tasks, specifically, the task of image classification. Building upon Agrawal et al. 2022, the proposed scheme combines the strengths of deep learning and Lagrange interpolation technique to mitigate the effect of straggling workers, and recovers approximate results with reasonable accuracy using outputs from any R out of N workers, where $R \leq N$. Our proposed scheme guarantees a minimum recovery threshold R for non-polynomial problems, which can be adjusted as a tunable parameter in the system. Moreover, unlike existing schemes, our scheme maintains flexibility with respect to worker availability and system design. We propose two system designs for our CC scheme that allows flexibility in distributing the computational load between the master and the workers based on the accessibility of input data. Our experimental results demonstrate the superiority of our scheme compared to the state-of-the-art CC schemes for image classification tasks, and pave the path for designing new schemes for distributed computation of any general ML classification tasks.

1. INTRODUCTION

Cloud platforms provide distributed computation services over many computing nodes for large-scale tasks. However, distributed computing systems are susceptible to straggler effects caused by slow or failing nodes, which can result in high latency in computation tasks. A naïve approach to address this issue is replication, which involves executing task copies on several computing nodes (workers) and returning the fastest-responding clones' results. However, this method incurs significant resource overhead. In this direction, a class of techniques known as coded computation (CC) has emerged as a prominent alternative. In CC schemes, redundancy is introduced into the input data based on the coding theory, and the encoded data is distributed to the workers for computation. This enables a master node to recover full computation results by decoding the results from only a few worker outputs, without waiting for all workers to return their results. Thus, such a scheme requires fewer resources to deal with the straggler effects compared to replication-based approaches [1].

Many existing CC schemes excel at recovering exact results on distributed tasks, but they are generally tailor-made for specific classes of functions, such as matrix multiplication [2, 3]. To provide a trade-off between accuracy and recover-threshold, some researchers have proposed CC schemes for approximate recovery of results [4, 5]. A notable example of CC scheme that covers a general class of functions, namely, polynomial functions, is the *Lagrange coded computation* (LCC) [6]. The LCC uses Lagrange's interpolation technique to guarantee exact recovery of results using outputs from a fixed minimum number of workers (known as the recovery threshold of scheme) [1]. However, all above-mentioned CC schemes are severely restricted in their application to several popular machine learning (ML) tasks, which typically incorporate deep neural networks (DNNs) with complex non-linear structures.

Several papers have proposed CC schemes for the training stage of deep neural networks (DNNs),[1] addressing the problem of stragglers as well as malicious attacks (seminal works include [7, 8]). For the inference stage of ML tasks, [9, 10] propose a CC scheme that "learns" an appropriate erasure code via a supervised deep learning technique. One approach jointly trains an auto-encoder for encoding and decoding operations, while the other trains a deep neural network (DNN) to perform the computation on the encoded data. However, they are limited to a single straggler and their performance degrades significantly with an increasing number of inputs.

In [11], the authors propose a CC scheme called AICC, that addresses the issues of exiting CC schemes by combining the powers of deep learning and Lagrange's interpolation technique, to provide approximate results in distributed computation tasks. The AICC is (1) applicable to general functions, including non-polynomials; (2) guarantee a recover-threshold that does not rely on the number of inputs; and (3) allows for a judicious trade-off between the accuracy and the computation load. However, as a proof-of-concept in [11], the AICC is applied only to certain non-polynomial matrix functions.

In this paper, we present a CC scheme based on AICC for the distributed computation of tasks pertaining to the inference phase of ML algorithms. Specifically, we apply the proposed CC scheme to an image classification task on the Fashion-MNIST dataset [12]. Unlike existing CC schemes for image classification [9, 10], our scheme allows for distribution of the tasks to any number of workers available, while still inheriting the recovery-through-interpolation property of LCC to guarantee the recovery of results from a fixed (but unspecified) subset of worker outputs. Furthermore, for applications with privacy concerns of the input data, we present a novel variant of AICC's computation operation that, unlike the scheme proposed in [11], only requires the encoded data, while showing only a small degradation in performance. Another promising characteristic of the proposed scheme is that most of its design parameters are tunable, making it amenable to the system's capabilities, and trade-off be-

JT and JZ acknowledge the support by the Australian Research Council under Project DE210101497 and the UoM-BUA Partnership scheme. NA is funded by the German Research Foundation (DFG) within their priority program SPP1914 "Cyber-Physical Networking", and SS acknowledges the support of joint project 6G-RIC with project identification numbers 16KISK020K and 16KISK030. (email: navneet.agrawal@tu-berlin.de)

[1] A supervised machine-learning algorithm consists of a *training* phase, where model parameters are trained using a dataset, and an *inference* or testing phase, where the pre-trained model is implemented over the new inputs, providing approximate results.

tween desired accuracy and available computation resources. The experimental results of this study verify that our scheme, using the same DNN architectures as used in [10], outperforms the accuracy achieved in [10] on the Fashion-MNIST dataset.

2. PRELIMINARIES

Consider a function f that maps any real-valued square matrix from $\mathcal{D} \subset \mathbb{R}^{M\times M}$ to an element of some finite dimensional Euclidean space $\mathcal{S}$. Note that the restriction of domain of f to square matrices is not a limitation of the proposed scheme, but a choice that reflects the application requirements, i.e. the images used in the classification task are represented as elements in $\mathcal{D}$. Define a set of K input matrices (called *dataset*) $\mathcal{X} := \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$, and the set of outputs of function f on the dataset $\mathcal{X}$ as $\mathcal{F} := \{f(\mathbf{X}_1), \ldots, f(\mathbf{X}_K)\}$. The objective is to obtain $\mathcal{F}$, given $\mathcal{X}$, with a minimum delay. The system consists of $N \geq K$ workers and a master node, which is responsible for collecting all the results in $\mathcal{F}$. To reduce the delay in obtaining the results, the computation task is distributed among the workers. A naïve approach is to distribute the K computations in $\mathcal{F}$ to any subset K of N workers, and wait for them to return their results to the master. Typically, such an approach is prone to unexpected delays due to the so-called *straggling effects* such as worker-node failure or slowdown. Moreover, some of the available worker resources are not used in this approach. The coded computation (CC) refers to a class of distributed computation schemes which use the available resources more efficiently, and can mitigate the straggling effects by using coding techniques to recover all desired results from a subset of worker outputs.

In general, the CC schemes operate by encoding the dataset $\mathcal{X}$ to generate N encoded data $\tilde{\mathbf{X}}_n, n = 1, \ldots, N$. The worker n performs some computation on the encoded data $\tilde{\mathbf{X}}_n$ and returns the result to the master. Then, the master recovers the desired results $\mathcal{F}$ upon receiving the computation results from any subset $R \leq N$ of workers. The smallest value of R for which the recovery of results can be guaranteed is called the *recovery threshold* of the scheme. In this way, the CC schemes can mitigate the impact of straggling.

The LCC [6] scheme is a CC scheme specifically designed only for polynomial functions f, say, of degree d. The functioning principle of the LCC can be summarized as follows (see [6] for details): The encoding operation amounts to the evaluation of a Lagrange polynomial at some distinct nonnegative scalars α_n for $n = 1, \ldots, N$. The Lagrange polynomial is of degree $K-1$, and its coefficients are linear functions of the input data $\mathcal{X}$, making it computationally inexpensive to implement. The worker n then computes the polynomial f on the encoded data $\tilde{\mathbf{X}}_n$. The computation result $f(\tilde{\mathbf{X}}_n)$ can be viewed as a composition of the Lagrange polynomial and the polynomial function f at a point α_n, and hence, it is a polynomial itself, of total degree $(K-1)d$. Hence, by using the Lagrange interpolation technique, this composite polynomial can be recovered (exactly) using $R = (K-1)d + 1$ results from the workers. The desired function values $\mathcal{F}$ can then be obtained by evaluating the polynomial at suitable points known to the master. Although LCC is optimal in terms of the recovery threshold, it suffers from two crucial limitations: (i) the function f is restricted to be a polynomial of the matrix-valued inputs; and (ii) the recovery threshold R of LCC grows proportionally with the number of inputs K and degree d of the polynomial f.

In [11], a learning-based approximate CC scheme is proposed, called AICC, to tackle the limitations of the LCC. The AICC scheme can be applied to a large class of functions, including non-polynomial functions, and provide approximate results which can be tuned to a desired accuracy based on the application requirements and system capabilities. The AICC scheme is shown in [11] to provide reasonably accurate results for certain matrix-valued functions, namely, computation of eigenvalues, dominant eigenvector, determinant, and exponential of a matrix, that are of interest in wireless communications.

The main objective of this paper is to extend the AICC scheme to functions that amount to evaluation of the inference stage of an ML algorithm. Specifically, we propose a CC scheme, based on AICC, for the problem of *image classification* over a given dataset. The ML algorithm for image classification task is structured so that most of its computations in the inference stage can be distributed to the workers. By carefully designing the encoding and computation operations such that their composition has a polynomial structure, we ensure that the desired results can be decoded from any subset of R worker results, as in the AICC scheme. It is worth noting that the image classification task considered in this paper requires different design approach compared to the approximate function computation tasks tackled in [11]. For example, the output of the function in image classification problem is the predictive probability of a certain class, while the target is one of the classes. Hence, the learning framework must be modified to support the problem and the available data. Moreover, the learning capabilities of DNNs involved in AICC can be enhanced by embedding the knowledge about the problem into design of DNNs.

As AICC is an essential part of this study, we dedicate Section 3 to its introduction. In Section 4, we formulate the image classification problem, provide details of the design choices for the DNNs involved in AICC encoder and computation operations, and describe the training procedure that achieves desired accuracy of results. The details of simulation, and the results comparing the proposed method with existing approaches, are provided in Section 5.

3. AI-AIDED CODED COMPUTATION (AICC)

In this section, we give a brief overview of the AICC scheme as proposed in [11]. Similar to the LCC scheme, the basic operations involved in the AICC scheme are *encoding*, *computation*, and *decoding* operations. The AICC enjoys the recover-through-interpolation property of the LCC, with guarantees of a recovery threshold that, in contrast to LCC, depends solely on tunable design parameters. At its core, the AICC scheme involves DNNs into its operations, whose learnable parameters are trained a priori using a training dataset. Indeed, the design and architecture of the DNNs in AICC influences its overall performance, and hence, they need to be carefully designed for a given problem. Most importantly, the recover-through-interpolation property of AICC rely on the polynomial structure of composition of AICC operations. This design aspect is highlighted in this section.

3.1. Encoding operation E

In CC schemes, encoding injects redundancy into the data before it is distributed to workers, aiming to obtain desired results with simple decoding. For example, LCC employs a linear function for encoding, resulting in a simple linear decoder [6]. Given the dataset $\mathcal{X} \in (\mathcal{D})^K$, the encoding operation in AICC is is a degree G polynomial $\mathsf{E} : \mathbb{R}_+ \to \mathbb{R}^{M\times M}$ given by $\mathsf{E}(\alpha) := \sum_{g=0}^{G} C_g \alpha^g$, where the coefficients $C_g \in \mathbb{R}^{M\times M}$, for $g = 0, \ldots, G$, are generated using the function $\Gamma_g : (\mathcal{D})^K \to \mathbb{R}^{M\times M}$, that involves pre-trained DNNs taking $\mathcal{X}$ as input. Each worker $n = 1, \ldots, N$ obtains the matrix $\tilde{\mathbf{X}}_n = \mathsf{E}(\alpha_n)$, where $\alpha_n > 0$ is a distinct for each worker n,

and it is known to the master for decoding. We remark that the coefficients (C_g), or equivalently the functions (Γ_g), are identical for all workers. It is also worth noting that DNNs involved in functions (Γ_g) can be arbitrary, as they do not affect the polynomial structure of the encoder E. Specific DNN designs for (Γ_g) for the image classification problem will be discussed in Section 4.

3.2. Computation operation H

The computation is performed by each worker n on the encoded data $\tilde{\mathbf{X}}_n := \mathsf{E}(\alpha_n)$, and the results are sent to the master as soon as they become available. The computation operation $\mathsf{H} : \mathbb{R}^{M\times M} \to \mathbb{R}^V$ is given by $\mathsf{H}(\mathbf{X}) := \sum_{p=0}^{P} \mathbf{V}_p \ \mathrm{Vec}(\mathbf{X}^p)$, where $\mathrm{Vec}(\mathbf{X})$ vectorizes the matrix $\mathbf{X}$ by stacking its columns, and the coefficients $\mathbf{V}_p := \Lambda_p(\mathcal{X})$, for $p = 0, \ldots, P$, are generated using the function $\Lambda_p : (\mathcal{D})^K \to \mathbb{R}^{V\times M^2}$. The output dimension V and the degree $P \in \mathbb{N}$ are design parameters.

3.3. Decoding operation

The decoding operation essentially involves the following two steps: First, the coefficients of the composite polynomial $\mathsf{D} := \mathsf{H} \circ \mathsf{E} : \mathbb{R}_+ \to \mathbb{R}^V$ of degree GP are obtained using the Lagrange interpolation technique via any $R = GP + 1$ input-output pairs from the set of worker outputs $\{(\alpha_1, \mathsf{D}(\alpha_1)), \ldots, (\alpha_N, \mathsf{D}(\alpha_N))\}$, where $R \leq N$ is one more than the degree of the polynomial D. Then, the polynomial D is evaluated at K distinct scalars $\beta_1, \ldots, \beta_K$ to obtain the desired results. Each scalar β_k, $k = 1, \ldots, K$, is chosen prior to the training procedure (as described in the following section), such that the output $\mathsf{D}(\beta_k)$ either corresponds to the approximate result $\hat{f}(\mathbf{X}_k) \approx f(\mathbf{X}_k)$ itself, or it can be easily transformed into one. An advantage of using the Lagrange's interpolation technique for the decoding operation is that both steps, the polynomial interpolation and the evaluation of results, involve only linear matrix operations, which are computationally inexpensive.

Remark 1 (Placement of operations). In systems where the dataset $\mathcal{X}$ is available to all worker (e.g. shared memory), the workers can essentially implement both the encoding and the computation operations, and send the results, along with corresponding α_n, to the master. In this case, the master only needs to perform the computationally inexpensive decoding operation. However, when the dataset $\mathcal{X}$ is not available to the workers, the master must also perform the encoding operation. In addition, the dataset $\mathcal{X}$ or coefficients $(\mathbf{V}_p)$ must be shared among the workers. Instead, to save communication and computation resources, we present a novel design of the computation operation H that only relies on the encoded data $\tilde{\mathbf{X}}_n$ (see Section 4.2.2). In Section 5, we present CC schemes for both cases described above, and compare the performance with different design parameters.

3.4. Training procedure

The parameters of DNNs involved in encoding and computation operations in the AICC scheme are trained such that, for all $k \in \{1, \ldots, K\}$, the cost of approximating the true value $f(\mathbf{X}_k)$ with the approximate output of the CC scheme $\hat{\boldsymbol{f}}_k := \mathsf{D}(\beta_k)$ over the training dataset is minimized. The distinct real-valued scalars β_k, for $k = 1, \ldots, K$, are chosen at the beginning, and they remain fixed during the entire training and test phases. In [11], the mean square loss function is used for the training. The training consists of forward and backward passes. In each forward pass, for each $k = 1, \ldots, K$, for every data pair $(\mathcal{X}, \beta_k)$, the output $\hat{\boldsymbol{f}}_k := \mathsf{D}(\beta_k) = \mathsf{H}(\mathsf{E}(\beta_k))$ is obtained. Then, in the backward pass, the parameters are updated based on the loss function between the output $\hat{\boldsymbol{f}}_k$ and the target value $f(\mathbf{X}_k)$.

4. CODED DISTRIBUTED IMAGE CLASSIFICATION

The DNN based ML approaches to the image classification problems are known to perform very well [13]. However, for most problems, the DNNs that provide sufficiently accurate results consist of a large number of parameters, and hence, their implementation requires significant computational resources. Moreover, in many applications, the image classification problem is required to be solved for numerous images at once, and with minimum latency. Therefore, distributed computation is often desirable in such applications to reduce latency.

In this section, we propose a CC scheme based on AICC that allows distributed computation of the image classification tasks. We describe the encoding, computation, and decoding operations in Sections 4.1, 4.2, and 4.3, respectively. In addition to ensuring the polynomial structure of the composite function $\mathsf{D} := \mathsf{H} \circ \mathsf{E}$ in our design, we also adopt DNN architectures that are known to enhance learning over image data. We compare our scheme with the state-of-the-art CC scheme proposed in [9] for such tasks, and present results from conventional (centralized) DNN-based approaches as a benchmark. We remark that, in addition to being superior in performance compared to [9], our scheme is inherently more flexible in terms of system requirements (i.e. accuracy and complexity of the algorithm, as well as recovery threshold, are design parameters), and guarantee a fixed recovery threshold for any number of workers.

4.1. Encoding operation design

The encoding operation follows the same structure as described in Section 3.1, i.e. E is a polynomial of degree G, with pre-trained coefficients $C_g := \Gamma_g(\mathcal{X})$, for all $g = 0, \ldots, G$. The two architectures employed for the DNNs in function Γ_g, for all $g = 0, \ldots, G$, are given in Table 1 (activation functions are omitted from the Table). One involves the multilayer perceptrons (MLPs), which makes use of fully-connected (FC) NN layers, and another is based on the convolution layers (CLs), which are known to perform well on image data. Both architectures use the ReLU activation functions in all but the final layer, and CLs use dilated convolution, as described in [14].

4.1.1. MLP for encoding polynomial E

For all $g = 0, \ldots, G$, the coefficient C_g in the encoding operation E is output of the function Γ_g. The function Γ_g is an MLP with the same architecture, but with different learned parameters. For MLP in Γ_g, the input is the KM^2 dimensional vector obtained by flattening each image in the dataset $\mathcal{X}$ and concatenating them together. The output of Γ_g (dimension M^2) is transformed into a $M \times M$ matrix to obtain the gth coefficient C_g of the encoding polynomial. Note that the learnable parameters in the MLP grow proportional to the size of the input KM^2, which could lead to overfitting in ML tasks [10].

4.1.2. Convolution layers (CLs) for encoding polynomial E

The CL has the advantage of processing the image in its original 2D structure, and hence, it can learn complex patterns in the image that are usually lost when the image is flattened. The CLs, representing the function Γ_g for $g = 0, \ldots, G$, takes K images in the dataset

Table 1. DNN architectures used in the proposed CC scheme.

MLP	CLs	Base-MLP
$KM^2 \times KM^2$	kern:3×3, dilation 1	$M^2 \times L_1$
$KM^2 \times M^2$	kern:3×3, dilation 1	$L_1 \times L_2$
	kern:3×3, dilation 2	$L_2 \times V$
	kern:3×3, dilation 4	
	kern:3×3, dilation 8	
	kern:3×3, dilation 1	
	kern:3×3, dilation 1	

$\mathcal{X}$ as K input channels in the first CL. In every CL, a kernel of dimension 3×3 is used with dilation (see [14]), and between each pair of CLs, there is a ReLU activation function (omitted in the table 1). The dilation increases the receptive field of the image without increasing parameters, captures features at multiple scales, and reduces spatial resolution loss compared to regular convolutions with larger filters [14]. Even though the DNN architecture with CLs requires more layers to combine all input pixels, it consists of fewer parameters compared to the MLP architecture, since the parameters are only required for defining the 3×3 kernel in a CL. This helps the CL architecture to be computationally inexpensive, and also avoid overfitting. Hence, employing CL architecture provides a better performance than MLP architecture on the image classification task, which is also confirmed via simulation in Section 5.

4.2. Computation operation design

For the computation operation H, as described in 3.2, we set the design parameter V as the number of classes in the image classification task. The worker computation H must be carefully designed to ensure that each component of its output $\mathsf{H}(\tilde{\mathbf{X}}) = \mathsf{H}(\mathsf{E}(\alpha)) \in \mathbb{R}^V$ is a polynomial in α. In light of remark 1, we propose two designs for the computation function H in the following: the first design H_S, as described in 3.2, requires the dataset $\mathcal{X}$ as input, and the second, H_B is designed to implemented without $\mathcal{X}$.

4.2.1. Computation function H_S

The computation function $\mathsf{H}_S : \mathbb{R}^{M\times M} \to \mathbb{R}^V$, illustrated in Figure 1, is defined as follows:

$$\mathsf{H}_S(\tilde{\mathbf{X}}) := \Omega\left(\sum_{p=0}^{P} \mathbf{V}_p \, (\tilde{\mathbf{X}}^{\odot p})\right), \tag{1}$$

where $P \in \mathbb{N}$ is a design parameter. The coefficients $\mathbf{V}_p \in \mathbb{R}^{M\times M}$, $p = 0, \ldots, P$, are obtained via the functions (involving DNNs) $\Lambda_p : (\mathcal{D})^K \to \mathbb{R}^{M\times M}$ such that $\mathbf{V}_p := \Lambda_p(\mathcal{X})$. The notation $\tilde{\mathbf{X}}^{\odot p}$ denotes the output of taking p times the Hadamard or element-wise product of the matrix $\tilde{\mathbf{X}}$ with itself. Note that the Hadamard product does not modify the polynomial structure of the output with respect to α in components of the encoded matrix $\tilde{\mathbf{X}}$, but raises the degree by p times. For the function Λ_p, we use the same MLP and CL architecture (including the activations) as in function Γ of the encoding operation, as shown in Table 1. The function Ω, defined as $\Omega : \mathbb{R}^{M\times M} \to \mathbb{R}^V$, has the Base-MLP architecture (with no activation functions) in Table 1 with parameters $L_1 = 200$ and $L_2 = 100$. In this design, it is assumed that either the dataset $\mathcal{X}$ is available to the workers, or the master node (with access to the dataset) computes coefficients $\mathbf{V}_p$ and broadcast them to the workers (coefficients $(\mathbf{V}_p)$ are the same for all workers).

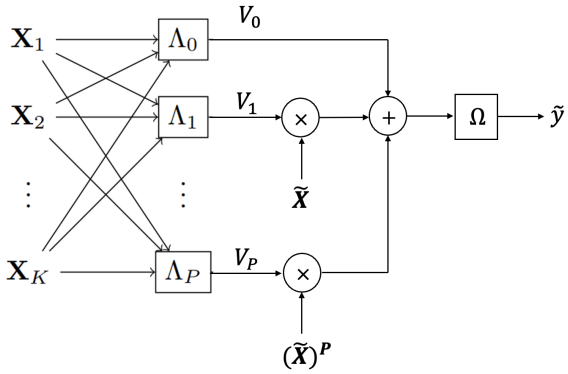

Fig. 1. The computation function H_S has a polynomial structure whose coefficients are functions of the input dataset $\mathcal{X}$. On the other hand, the function H_B has a linear structure, and it does not depend on the input dataset $\mathcal{X}$.

4.2.2. Computation function H_B

The computation function H_B is designed to only depend on the inputs $\tilde{\mathbf{X}}$, and hence, workers implementing H_B do not need access to the dataset $\mathcal{X}$ or any other information from the master, except the encoded input $\tilde{\mathbf{X}}$. The function $\mathsf{H}_B : \mathbb{R}^{M\times M} \to \mathbb{R}^V$ applies a sequence of linear transformations on the input matrix $\tilde{\mathbf{X}}$, keeping the polynomial structure of the composite function $\mathsf{D} := \mathsf{H} \circ \mathsf{E}$. The linear operator H_B is essentially a cascade of matrix multiplications on the input $\tilde{\mathbf{X}}$, and it can be seen as applying a DNN (MLP or CNN), without any non-linear activation functions, to the input $\tilde{\mathbf{X}}$. In other words, the function H_B simply applies some linear matrix operations to the input $\tilde{\mathbf{X}}$, which does not change the degree of the composite polynomial.

We employ two DNNs architectures for the function H_B as shown in TABLE 1: (1) *MLP*-H_B: the Base-MLP only and (2) *CNN*-H_B: CLs followed by the Base-MLP. Note that both architectures MLP-H_B and CNN-H_B are without any activation functions.

Figure 2 illustrates the general data transformation process of our approach during the encoding and computation operations, where the function H could be either H_S or H_B. The dotted line connecting the input dataset $\mathcal{X}$ and function H represents the two design strategies H_S and H_B, indicating whether the input dataset $\mathcal{X}$ is used in the computation operation or not. As a baseline for comparison, we use two *centralized* DNN based algorithms for the image classification task: (1) *MLP-baseline*, and (2) *CNN-baseline*, by respectively adding ReLU activation function between each pair of layers in MLP-H_B and CLs of the CNN-H_B. Note that the baseline algorithm is not distributable, and does not apply any encoding or decoding operation.

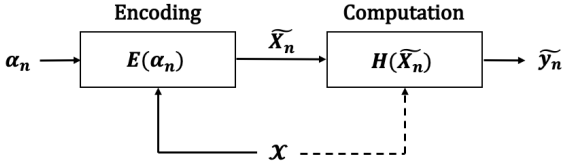

Fig. 2. Encoding and Computation process

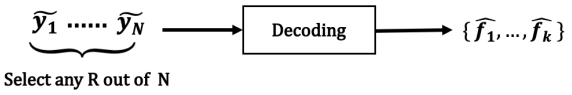

Fig. 3. Decoding process

4.3. Decoding operation design

The decoding operation is essentially the same as described in Section 3.3, except that after recovering, we apply a *Softmax* operation to convert the vector $\tilde{\boldsymbol{y}} \in \mathbb{R}^V$ to a probability mass function for each of the V classes. The recovery is ensured by the polynomial structure of the composite polynomial $\mathsf{D} := \mathsf{H} \circ \mathsf{E}$. Using computation operation H_S or H_B results in the polynomial D of degree GP or G, respectively. Thus, the recovery threshold for the schemes corresponding to H_S and H_B is $R = GP+1$ and $R = G+1$, respectively. After interpolating the polynomial D, we apply the *Softmax* function to the result $\mathsf{D}(\beta_k)$ to obtain $\acute{\boldsymbol{y}}_k := \text{Softmax}(\mathsf{D}(\beta_k))$, and select the index with the highest probability as the prediction label L_k for the input image $\mathbf{X}_k$.

4.4. Training procedure

We follow the same training procedure as described in Section 3.4, except that the loss function is different in our scheme. In the training dataset, we are given multiple data with the inputs $\mathcal{X} := \{\mathbf{X}_1, \ldots, \mathbf{X}_K\}$ and corresponding true labels $f(\mathbf{X}_k) \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_V\}$, for each $v = 1, \ldots, V$, where $\boldsymbol{e}_v \in \mathbb{R}^V$ is the one-hot vector with all elements zero except one at position v. At each forward pass of the scheme with input β_k, where $(\beta_1, \ldots, \beta_K)$ are distinct scalars that are fixed, we obtain $\acute{\boldsymbol{y}}_k := \text{Softmax}(\mathsf{D}(\beta_k))$ via the encoding, computation, and decoding operations, as described above. The backward pass updates the DNN parameters using an iterative algorithm based on the *cross entropy loss function* between the true labels $f(\mathbf{X}_k)$ and the estimated probability mass $\acute{\boldsymbol{y}}_k$.

5. SIMULATIONS

We implemented our learning-based CC model using PyTorch, which enabled us to experiment with different DNN architectures for the encoding and computation operations. We use the Fashion-MNIST dataset [12] for the image classification task, which has images of dimension $\mathbb{R}^{M \times M}$ with $M = 28$, and $V = 10$ target classes. We train our model for 20 epochs in two settings: (1) a batch size of 64 samples with $K = 2$ inputs in dataset $\mathcal{X}$, and (2) a batch size of 32 samples with $K = 4$ inputs in dataset $\mathcal{X}$. Each minibatch sample has K images drawn randomly without replacement, and no image was sampled more than once per epoch. We used the Adam optimizer with a learning rate of 0.001 to update the DNN parameters. The scalar values $\{\beta_1, \beta_2, \ldots, \beta_K\}$ are fixed to $\beta_k = k/K$ for $k = 1, \ldots, K$ throughout the experiment. During the testing phase, we fix the scalar values $\{\alpha_1, \alpha_2, \ldots, \alpha_N\}$ to $\alpha_n = n/(N+1)$ for $n = 1, 2, \ldots, N$. We evaluate the performance of our model based on its accuracy in reconstructing the results on a separate test dataset. The *evaluation criteria* is the *accuracy* of predictions on the test dataset, defined as the proportion of correctly predicted labels to the total number of images in the test dataset.

5.1. Results

In the following, we denote the proposed CC schemes with the computation functions H_B and H_S as H_B and H_S, respectively. We compare the performance of our scheme in Section 5.1.1 for different DNN architectures used in encoding and computation operations, as well as with [10] and centralized baseline models. Furthermore, in Section 5.1.2, we investigate our scheme for different selections of encoder and computation polynomial degrees G and P, respectively, and discuss how one can select these free parameters in our scheme to make a judicious trade-off between accuracy and computational complexity.

5.1.1. Comparison of different DNN architectures

For fair comparison, in our simulation, we use the architecture of MLPencoder and CNNencoder from [10] for our encoding and computation operations (except that in H_B activations are not implemented). Table 2 presents results for a system (the same as in [10]) with workers $N = 3$, input size $K = 2$ and recovery threshold $R = 2$. Note that, in both our schemes H_B and H_S, to ensure the recovery threshold $R = 2$, we use the degrees $(G = 1, P = 1)$, respectively, for the encoding and computation operations. Our CC scheme outperforms the schemes presented in [10] in terms of accuracy, regardless of whether we use H_B or H_S. Moreover, the encoder using CL architecture (see Section 4.1.2) outperforms the MLP encoder in both models. Compared to the baselines, the degradation in accuracy of our scheme based on MLP and CNN encoders is around 5% and 7%, respectively, which is an acceptable degradation considering the fact that the baseline algorithm cannot be distributed. Most notably, these results demonstrate that CL architecture performs better than MLP, concurring with the discussion in Section 4.1.2.

Table 2. Evaluating accuracy of schemes for $R = 2$ and $K = 2$.

Enc+Comp	H_B	H_S	[10] model	Baseline
MLP+MLP	**83.93**	**84.34**	81.96	89.12
CNN+MLP	**85.81**	**84.52**	82.53	
MLP+CNN	84.11	85.37	–[2]	92.01
CNN+CNN	85.94	86.62	–[2]	

5.1.2. Performance trade-off in selection of design parameters

We conduct additional experiments to evaluate the impact of two parameters on the performance of our two models, namely, the degree of composite function $\mathsf{D} := \mathsf{H} \circ \mathsf{E}$, denoted by D in the following, and input size K. Table 3 shows results for computation operations H_S and H_B, with $D = GP$ and $D = G$, respectively, where we have implemented CL architecture for both operations (see Section 4 for details). The column '# of params' denotes the total number of learnable parameters (scaled such that $100\% \equiv 2218197$) in all the DNNs involved in our scheme, and it is one of the indicators of computational complexity of the inference using such model. Within the same parameter setting for K and R, the model H_S performs better than the model H_B, as the former benefit from the nonlinear activations involved in coefficients $\mathbf{V}_p := \Lambda_p(\mathcal{X})$, for $p = 0, \ldots, P$, of the computation operation. Moreover, the accuracy gap between

[2]Some of the results of [10] cannot be compared directly with our scheme because they correspond to the *ResNet18* model in the computation operation, while our schemes H_B and H_S uses the CL architecture presented in Table 1 (also see Section 4). Nonetheless, even though the *ResNet18* model is specifically designed CNN architecture that is well-trained for this problem, our schemes, with a simple training procedure and design, only show $2-4\%$ degradation in performance w.r.t. [10].

the two models widens as the number of inputs K in the dataset increases, which in turn provides more parameters for learning to H_S compared to H_B.

For a given K, increasing the value of D enhances the accuracy, which suggests that one can improve model performance by tuning the parameters G and/or P. Also, increasing the value of P in the computation function H_S results in a better performance. For instance, for $K = 4$ and $D = 4$, the accuracy of the model H_S is 75.6% with $G = 4$ and $P = 1$, but it improves to 82.3% with $G = 1$ and $P = 4$, even though both models have the same number of parameters. In addition to the improved accuracy, for schemes with the same recovery threshold R, a lower value of G (or higher value of P) means less computation for the encoding operation, which is usually implemented by the master node. Hence, for model H_S, by increasing the degree of computation operation and reducing the degree of encoding operation, we can reduce the overhead of the master as well as improve the accuracy. However, to improve the accuracy of the model H_B, we can only increase the value of G in the encoding function E, which would increase the workload of the master. Nonetheless, the model H_B does not require sharing input data $\mathcal{X}$ with each worker, which enhances data privacy and minimizes network communication. Hence, our scheme provides flexibility of the choice of model, based on the specific system requirements and capabilities.

Table 3. Varying degrees G, P and number of inputs K.

K	R	H_B/H_S	G, P	accuracy	# of params
2	3	H_B	2, 1	**87.3**	21 %
		H_S	2, 1	88.5	23 %
			1, 2	**88.7**	23 %
4	3	H_B	2, 1	**71.2**	50 %
		H_S	2, 1	71.3	75 %
			1, 2	**76.8**	75 %
	5	H_B	4, 1	**75.6**	80 %
		H_S	4, 1	75.6	100 %
			2, 2	78.0	86 %
			1, 4	**82.3**	100 %

6. CONCLUSION

In conclusion, this paper proposes a novel coded computation scheme for distributed computation of inference phase of machine learning algorithms based on AICC [11]. The proposed scheme allows for the distribution of tasks to any number of workers while ensuring the recovery-through-interpolation property of LCC for an approximate recover of results with desired accuracy. Moreover, a novel variant of AICC's computation operation is presented to address privacy concerns of input data, with only a small degradation in performance. The proposed scheme's tunable design parameters make it amenable to systems with varying capabilities, and allows for a judicious trade-off between accuracy and available computation resources. Experimental results for image classification on the Fashion-MNIST dataset demonstrate that the proposed scheme is superior (in terms of accuracy) and more flexible (in adaptation to various systems) than the existing schemes.

7. REFERENCES

[1] Songze Li and Salman Avestimehr, "Coded computing: Mitigating fundamental bottlenecks in large-scale distributed computing and machine learning," *Foundations and Trends in Communications and Information Theory*, vol. 17, no. 1, pp. 66–96, 2020.

[2] Qian Yu, Mohammad Maddah-Ali, and Salman Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[3] Sanghamitra Dutta, Mohammad Fahim, Farzin Haddadpour, Haewon Jeong, Viveck Cadambe, and Pulkit Grover, "On the optimal recovery threshold of coded matrix multiplication," *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 278–301, 2019.

[4] Vipul Gupta, Shusen Wang, Thomas Courtade, and Kannan Ramchandran, "Oversketch: Approximate matrix multiplication for the cloud," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 298–304.

[5] Tayyebeh Jahani-Nezhad and Mohammad Ali Maddah-Ali, "Codedsketch: A coding scheme for distributed computation of approximated matrix multiplication," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 4185–4196, 2021.

[6] Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 2019, pp. 1215–1225.

[7] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos, "Draco: Byzantine-resilient distributed training via redundant gradients," in *International Conference on Machine Learning*. PMLR, 2018, pp. 903–912.

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

[9] Jack Kosaian, KV Rashmi, and Shivaram Venkataraman, "Learning a code: Machine learning for approximate nonlinear coded computation," *arXiv preprint arXiv:1806.01259*, 2018.

[10] Jack Kosaian, KV Rashmi, and Shivaram Venkataraman, "Learning-based coded computation," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 227–236, 2020.

[11] Navneet Agrawal, Yuqin Qiu, Matthias Frey, Igor Bjelakovic, Setareh Maghsudi, Slawomir Stanczak, and Jingge Zhu, "A learning-based approach to approximate coded computation," in *2022 IEEE Information Theory Workshop (ITW)*. IEEE, 2022, pp. 600–605.

[12] Han Xiao, Kashif Rasul, and Roland Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[13] Shivam S Kadam, Amol C Adamuthe, and Ashwini B Patil, "Cnn model for image classification on mnist and fashion-mnist dataset," *Journal of scientific research*, vol. 64, no. 2, pp. 374–384, 2020.

[14] Fisher Yu and Vladlen Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.