Lagrange Coding for Tensor Network Contraction: Achieving Polynomial Recovery Thresholds

Kerong Wang, **Zheng Zhang**, and **Qian Yu**
Department of Electrical and Computer Engineering
University of California, Santa Barbara
Santa Barbara, CA, USA
Email: kerong@ucsb.edu, zhengzhang@ece.ucsb.edu, qianyu02@ece.ucsb.edu

***Abstract*—We study the problem of distributedly computing a general class of computational tasks, referred to as *multilinear gradient computation*. Multilinear gradient computation evaluates a multilinear map of gradient-type functions, which computes the sum of 'sub-gradients,' defined as homogeneous polynomials of subsets of the input. Multilinear gradient computation is a generalization of gradient-type computation and has many applications in areas including machine learning and quantum computing. We propose Generalized Lagrange Codes (GLC), which universally compute *any* multilinear gradient computation. We apply GLC to establish a computing scheme for parallel tensor network contraction, named Lagrange Coded Tensor Network Contraction (LCTN), and show that LCTN achieves a recovery threshold with polynomial dependency on the parameters of the tensor networks. This provides a strict and exponential improvement upon the best-known results.**

I. Introduction

As modern large-scale computations are carried out under distributed schemes, resiliency against stragglers, adversaries, and possibly colluded workers has emerged as a major challenge [1]–[5]. Coded computing algorithms have been developed to tackle these challenges effectively [1], [3], [6]–[11]. A coded computing system consists of a master node and several worker nodes. The master first encodes the given dataset by taking a linear combination of the original data and distributes the coded version to each worker. Each worker then performs a pre-determined computational task on the assigned data and returns the result to the master. Finally, the master retrieves the desired output by decoding the workers' outputs. There have been numerous coded computing designs for specific computational tasks, including polynomial evaluation [7], large-scale matrix multiplication [3], [12]–[14], gradient-type computation [2], etc. The minimum number of successful workers' outputs required in the worst case for decoding is called the *recovery threshold.* Designing coded computing algorithms with low recovery thresholds has become a problem of considerable interest.

Tensors are mathematical objects that can be used to represent high-dimensional data. Tensor networks are building blocks in many fields, including modern quantum computing, machine learning, robotics, and control systems, etc. [15]–[22] Several computing designs for efficient tensor network contraction have been developed [18], [23], [24].

There have been studies that proposed coded computing designs for straggler-resilient tensor network contraction. However, there is an exponential gap between the achieved recovery threshold and the anticipated lower bound. We propose a straggler-resilient coding design that works for a larger class of computational tasks, i.e., multilinear gradient computation; tensor network contraction is a special case within the class and a direct application of our design. We will show that our coding design achieves a recovery threshold that is exponentially better than the best-known result proposed in [18]. Moreover, our design can be applied to any task within the same class, such as ordinary gradient-type computation [2].

A. Polynomial Coded Computing (PCC) and Lagrange Coded Computing (LCC)

We first provide a preliminary overview regarding prior works in coded computing. *Polynomial Coded Computing* (PCC) is a general framework for designing computational schemes. It was first introduced in [3] and formally described in [1]. Since its introduction, PCC has been widely adopted in the field of resilient and secure computing design [7], [14], [25], [26]. Given a dataset X and the task of evaluating a function $f(X)$ over a cluster of distributed workers, PCC encodes the input with a carefully designed vector-valued polynomial u and assigns each worker its evaluation at a distinct point. Then, if the workers can compute over the assigned value with a function h, the master can recover the coefficients of the composed function $h \circ u$ from a subset of successful worker outputs using polynomial interpolation. Therefore, by designing a function u to ensure the recoverability of $f(X)$ from the composed polynomial, we obtain computation schemes that are resilient to stragglers.

As an application of polynomial code to multivariate polynomial evaluation, *Lagrange Coded Computing* (LCC) was introduced in [7]. It achieves optimal resiliency against stragglers, security against adversaries, and privacy against collusion of workers. The coding idea of LCC has also been popularized for its numerous applications in resilient, secure, and privacy-preserving computing [27]–[32]. Consider an N-worker system (the computing system that contains a master and N workers) with a goal of computing $f(X_i)$ for all X_i in a large dataset $X = (X_1, \ldots, X_k)$, where f is an arbitrary multivariate polynomial. LCC encodes the dataset using Lagrange polynomials to obtain N coded version of

the dataset, denoted by $\tilde{X_1}, \ldots, \tilde{X_N}$. More specifically, each worker i would receive

$$\tilde{X_i} = u(z_i) \triangleq \sum_{j\in[n]} X_j \cdot \prod_{k\in[n]\setminus\{j\}} \frac{z_i - x_k}{x_j - x_k},$$

where $z_1, \ldots, z_N$, and $x_1, \ldots, x_n$ are some arbitrary distinct elements in a certain base field $\mathbb{F}$. Each worker i would then evaluate $f(\tilde{X_i})$, as if no coding had taken place. Finally, the master decodes the desired output by a subset of successful worker outputs using polynomial interpolation.

B. Notations

Throughout the paper, for any $n \in \mathbb{N}^+$, we use $[n]$ to denote the set of integers $\{1, 2, \ldots, n\}$. We denote the total degree [1] of a polynomial f by $\deg f$.

II. Definitions and Problem Formulation

We provide a formal definition of multilinear gradient computation. Then, we show that tensor network contraction can be viewed as a special case of multilinear gradient computation.

A. Multilinear Gradient Computation

As introduced in [2], a function G is *gradient-type* and *characterized by* a multivariate polynomial g if it satisfies $G(X) = g(X^{(1)}) + \cdots + g(X^{(L)})$ for some partition of the input X. Given a dataset $X = (X^{\{1\}}, \ldots, X^{\{n\}})$, the goal of *multilinear gradient computation* is to compute

$$f(X) := \Gamma\left(G_1(X^{\{1\}}), \ldots, G_n(X^{\{n\}})\right),$$

where Γ is a multilinear map and each G_k is a gradient-type function. Specifically, for any $k \in [n]$, we assume that $X^{\{k\}}$ is partitioned into $(X^{\{k\}(1)}, \ldots, X^{\{k\}(L_k)})$, so that $G_k(X^{\{k\}}) = g_k(X^{\{k\}(1)}) + \cdots + g_k(X^{\{k\}(L_k)})$ for a homogeneous multivariate polynomial g_k.

As illustrated in Fig. 1, we consider an N-worker system where (coded) computation inputs are assigned to the workers. The i-th worker would receive $(\tilde{X}_i^{\{1\}}, \ldots, \tilde{X}_i^{\{n\}}) := (u_1(z_i), \ldots, u_n(z_i))$ for some fixed $z_i \in \mathbb{F}$. Here $u_1, \ldots, u_n$ are polynomials with vector-valued coefficients, the polynomial's coefficients have the same size as the subpartitions of the corresponding input entries. In particular, the coefficients of u_k match the size of any $X^{\{k\}(\ell)}$. We are often interested in encoding functions with low complexities, e.g., the linear codes where the coefficients u_k's are linear combinations of the subpartitions of $X^{\{k\}}$.

The workers should compute the multilinear map of partial gradients, i.e., $\Gamma(g_1(\tilde{X}_i^{\{1\}}), \ldots, g_n(\tilde{X}_i^{\{n\}}))$. Then, the master uses a decoding function that takes a subset of successful workers' results as inputs to recover the final result. Multilinear gradient computation has appeared in several computing tasks, including parallel optimization [33], numerical PDE [34], and tensor algebra [35]. For simplicity, we ignore the trivial cases of affine gradient functions (i.e., where any G_k has a degree no greater than 1) as they can be precomputed before the tasks are assigned to the workers. We also ignore the trivial case of $L_k = 1$ for any k, i.e., $G_k\left(X^{\{k\}}\right) = g_k\left(X^{\{k\}(1)}\right)$, as no encoding is required for such components.

[1] We adopt the definition from [7]. The *total degree* of a polynomial f is the maximum among all the total degrees of its monomials. When discussing finite $\mathbb{F}$, we resort to the canonical representation of polynomials, in which the individual degree within each term is no more than $(|\mathbb{F}| - 1)$.

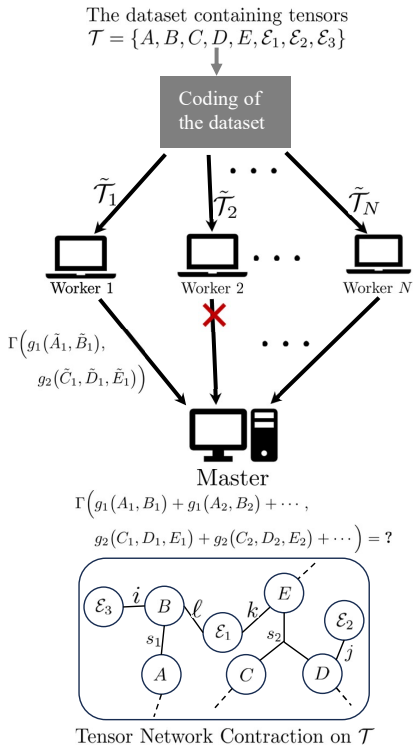

Fig. 1: An illustration of the system model for a specific multilinear gradient computation task arising from a tensor network contraction. In this case, A, B are contracted over the index s_1, and C, D, E are contracted over the index s_2. The master distributes the task by partitioning and slicing these tensors along corresponding indices. Thus, the task becomes a linear combination of contractions over the same tensor network but with certain tensors replaced by their partitions, whose sizes are fractions of the original input size. By treating tensor network contraction as a multilinear gradient computation, each worker is assigned a coded version of the tensors and computes a task where the subgradients g_1 and g_2 are the individual contractions over s_1, s_2, and the multilinear map Γ being the rest of the contractions. With a carefully designed coding strategy, the master can decode the required result even under the presence of *stragglers* (in this case worker 2).

B. Parallel Tensor Network Contraction

We study tensor network contraction as an example of multilinear gradient computation. We provide a concise overview, and we refer our readers to [36]–[40] for a more extensive review of these subjects.

Definition 1. *For a given base field $\mathbb{F}$, a tensor T is a multidimensional array with each entry chosen from $\mathbb{F}$. The*

order *of a tensor is the number of indices. Thus, an order-n tensor can be written as $T_{s_1,\ldots,s_n}$, where $s_1,\ldots,s_n$ are the indices of T.*

For example, a 3×3 matrix is an order-2 tensor, and each index has dimension 3. Consider a set of tensors $\mathcal{T}$ and a subset of their indices $\mathcal{S}$ sharing the same dimension. We use $T^{\{s\}}$ to denote the tensor obtained by slicing any $T\in\mathcal{T}$ over indices in $\mathcal{S}$, i.e., $T^{\{s\}}$ is the tensor of lower order where indices of T in $\mathcal{S}$ are fixed to the value of s.

Definition 2. *An single-index contraction on $\mathcal{T}=\{T_1,\ldots,T_n\}$ over an index set $\mathcal{S}$ is defined as*

$$f_{\mathcal{S}}(\mathcal{T})=\sum_{s}\left(\bigotimes_{k=1}^{n}T_k^{\{s\}}\right),$$

where '$\bigotimes$' denotes the tensor product. A multi-index contraction is defined as the composition over a sequence of single-index contractions.

The single-index contraction can be seen as a generalization of the inner product of vectors, where we take elementwise products of tensors and compute partial sums as the corresponding entry of the output tensor. The set $\mathcal{S}$ is often represented as a *hyperedge* connecting vertices that represent individual tensors. Since the ordering of single-index contraction does not affect the outcome of their composition, multi-index contraction is often depicted as a tensor network, where the disjoint sets of indices in each contraction are represented as unordered individual hyper-edges. Thus, multi-index contraction is also referred to as the tensor network contraction.

Example 3. *The matrix product between A and B is a single-index contraction that yields a matrix C where $C_{i,k}=\sum_j A_{i,j}B_{j,k}=\sum_j A^{\{j\}}\bigotimes B^{\{j\}}$. The chain matrix product over $A_1,A_2,\ldots,A_n$ is a multi-index contraction, where the individual single-index contractions are the matrix products of the adjacent matrices.*

In this paper, we consider the parallel tensor network contraction formulated in [18], where the overall computation task is viewed as a two-stage procedure. In the first stage, a subset of single-index contractions is performed, where no indices from distinct contractions are from the same tensor, i.e., there are no adjacent indices across different edges in the network representation. The second stage completes the remaining contractions. Specifically, if we denote the number of contractions in the first stage by n and the number of tensors involved in each contraction by $m_1,\ldots,m_n$,[2] a tensor network contraction under this framework can be written as a function f taking a set of tensors $\mathcal{T}=\{T^{\{i\}(j)}\}_{i\in[n],j\in[m_i]}\cup\{\mathcal{E}\}$ as the input and returns the following tensor

$$f(\mathcal{T})=\sum_{-}\bigotimes_{i=1}^{n}\left(\sum_{s_i}\bigotimes_{j=1}^{m_i}T_{s_i,-}^{\{i\}(j)}\right)\mathcal{E}_{-},\tag{1}$$

[2]For simplicity, we assume the non-trivial case of $m_i\geq 2$.

where the summation over non-adjacent indices $s_1,\ldots,s_n$ denotes the contractions in the first stage, and the summation over '$-$' denotes the contractions in the second stage.

The overall contraction operator f can be written as the summation of network contractions, where each $T^{\{i\}(j)}$ is replaced by its sliced tensor over index s_i, summed over all possible values of the indices $s_1,\ldots,s_n$. Parallel tensor network contraction distributes the computation task by assigning each worker the same task of computing one individual term of this summation, except that the inputs are replaced with coded versions of each $T^{\{i\}(j)}$ while the computation proceeds as if the inputs are uncoded sliced tensors.

The parallel tensor network contraction can be viewed as a special case of multilinear gradient computation. Specifically, the overall network contraction $f(\mathcal{T})$ can be written as the composition of a multilinear function and gradient-type computations by setting $X^{\{i\}}=\{T^{\{i\}(j)}\}_{j\in[m_i]}$, where each individual G_i is the single-index contraction over index s_i, and Γ is the composition for the rest of the contractions. Hence, if any result can be obtained for general multilinear gradient functions, they can be directly applied to parallel tensor network contraction.

C. Problem Formulation and Prior Works

Our goal in this paper is to design straggler-resilient coded computing schemes for multilinear gradient computation. Specifically, we aim to design the encoding functions to minimize the recovery threshold for clusters with arbitrarily many workers, but without increasing their computational loads. As a common assumption in coding theory, we focus on the important case where the base field $\mathbb{F}$ is sufficiently large. We also aim to achieve similar straggler resilience for parallel tensor network contraction.

Prior Works: There have been several studies on efficient algorithms for large tensor network contraction [18], [41]–[43]. In [18], Lee et al. introduced the first implementation of polynomial-based coding designs for tensor network contraction. They provided coding designs for two scenarios. 1) For a tensor network contraction over n non-adjacent indices, where each index i has dimension L_i and is connected with two tensors, i.e. $m_i=2$ for all $i\in[n]$, they provided the 2-node code by composing the polynomial code [3] and Matdot [14], which achieves recovery threshold of $2\prod_{i=1}^{n}L_i-1$. 2) Under the same setup while each m_i is an arbitrary integer, they proposed the hyperedge code, which achieves a recovery threshold of $\prod_{i=1}^{n}\frac{(m_i^{L_i}-1)}{m_i-1}$. To the best of our knowledge, their work is the first attempt to apply coded computing to achieve reliable and secure tensor network contraction. However, their design for the general case incurs an exponential cost in recovery threshold as the number of tensors involved in any single contraction increases.

III. Main Results

We proposed the *generalized Lagrange coding* to achieve the following results.

Theorem 4. *Under the setup of II-A, for any arbitrary multilinear gradient function f characterized by n gradient-type functions $G_1, \ldots, G_n$, and for any fixed partitioning parameters $L_1, \ldots, L_n$, there exists a computing scheme that achieves a recovery threshold of*

$$R \leq \left(2 + \sum_{j:\deg G_j > 2} \deg G_j \left(\prod_{k:\deg G_k > 2} L_k - 1\right)\right) \cdot \prod_{k:\deg G_k = 2} L_k - 1.$$

Corollary 5. *For any tensor network contraction where non-adjacent indices $s_1, \ldots, s_n$ are contracted, where each index s_i has dimension L_i and is connected to m_i tensors. There exists a straggler-resilient coding design that achieves a recovery threshold of*

$$R \leq \left(2 + \sum_{j:m_j > 2} m_j \left(\prod_{k:m_k > 2} L_k - 1\right)\right) \cdot \prod_{k:m_k = 2} L_k - 1.$$

Remark 1. Similar to LCC, our proposed coding construction depends only on the degrees of the computational task rather than their identities. Therefore, they *universally* compute general multilinear gradient functions and enable pre-encoding without knowing the identity of the computation task.

Remark 2. As a comparison with the results in [18], under the special case $m_i = 2$ for all $i \in [n]$, our algorithm achieves the same recovery threshold as their 2-node code, but with a different coding construction that applies beyond homogeneous polynomial function. For general m_i, our algorithm provides a strict and exponential improvement. However, we shall highlight that our algorithm can be applied to the class of all multilinear gradient functions.

Remark 3. As a potential alternative, one may directly apply LCC by viewing the computation task as the summation for the evaluations of $\Gamma(g_1, \ldots, g_n)$ over all $\prod_i L_i$ possible combinations of the uncoded input segments. This naive application achieves a recovery threshold of $(\sum_k \deg G_k)(\prod_k L_k - 1) + 1$. Our generalized Lagrange coding also provides a strict and exponential improvement (See Fig. 2 for example).

Remark 4. Although our coding design provides one-shot computation designs for arbitrary tensor network contraction, practical implementations could benefit from preprocessing a subset of the contractions (instead of applying our scheme directly to the entire operation) to reduce the recovery threshold. The preprocessing could be carried out with either the proposed generalized Lagrange coding or matrix codes such as Entangled polynomial codes [9], [44] for the two-tensor contractions case. This multi-stage approach allows a flexible tradeoff between the straggler resilience and the number of rounds in communication and computation.

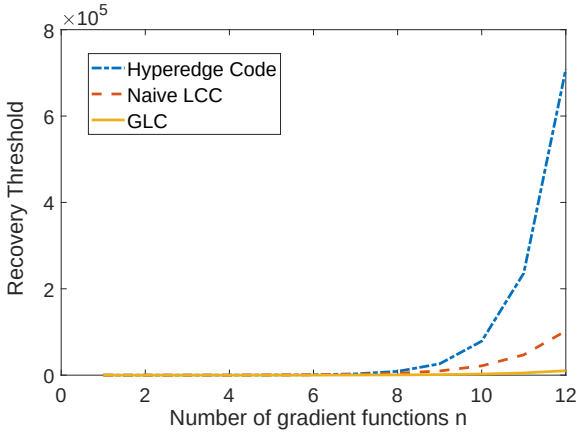

Fig. 2: Comparison of the recovery threshold achieved by GLC and others (Hyperedge code [18] and LCC [7]). For illustrative purpose, we fix $L_k = 2$ for all k, $m_i = 2$ for all $i \in [n-1]$ and $m_n = 3$. GLC significantly outperforms others.

IV. Lagrange Coded Tensor Network Contraction

A. Illustrating Example

We start with an illustrating example of tensor network contraction. Consider $\mathcal{T} = \{A, B, C, D, E\}$, where the individual tensors are over a base field $\mathbb{F}_{17}$. We are contracting on indices i and j, where tensors A, B are connected to i; tensors C, D, and E are connected j. We assume that both indices have a dimension of 2, i.e., $L_1 = L_2 = 2$. We want to compute $f(\mathcal{T}) = \sum_{i\in[2], j\in[2]} \sum_{-} A_{i,-} B_{i,-} C_{j,-} D_{j,-} E_{j,-}$, where the summation over $-$ represents a second-stage contraction.

Assuming the contraction is carried out over a 16-worker system. We first compare our achievable results with existing or preliminary approaches. First, the uncoded replication will group the works into 4 subclasses to independently carry out the four individual products, $\sum_{-} A_{i,-} B_{i,-} C_{j,-} D_{j,-} E_{j,-}$ for each possible pair of values of the indices (i, j). This replication approach achieves a recovery threshold of 13 since at most 3 stragglers can be tolerated in the worst case. On the other hand, the hyperedge code in [18] achieves a recovery threshold of 12. Our code achieves a recovery threshold of 9, which is strictly lower than both approaches.

Our proposed LCTN follows the PCC approach that encodes the input tensors with the following polynomials: For $T \in \{A, B\}$, we define

$$u_T(z) := \eta_1 T_{1,-}(z - \alpha_1) + \eta_2 T_{2,-}(z - \alpha_2);$$

and for $T \in \{C, D, E\}$, we let

$$v_T(z) := \omega_1 T_{1,-} (z - \beta_{2,1}) (z - \beta_{2,2}) + \omega_2 T_{2,-} (z - \beta_{1,1}) (z - \beta_{1,2}).$$

In this example, we fix $\alpha_1 = 0$ and $\alpha_2 = 1$. We let $\beta_{1,1} = 2, \beta_{1,2} = 6, \beta_{2,1} = 0$, and $\beta_{2,2} = 1$. Then, we choose $\eta_1 = \eta_2 = \omega_1 = \omega_2 = 1$.

Recall that in the PCC framework, each worker $k \in [N]$ takes the evaluation of each individual encoding polynomial at

a distinct point $z_k \in \mathbb{F}$, and receives $(\tilde{A}_k, \tilde{B}_k, \tilde{C}_k, \tilde{D}_k, \tilde{E}_k) \triangleq (u_A(z_k), u_B(z_k), v_C(z_k), v_D(z_k), v_E(z_k))$. Then, the worker computes $\sum_{-} \tilde{A}_{k,-}\tilde{B}_{k,-}\tilde{C}_{k,-}\tilde{D}_{k,-}\tilde{E}_{k,-}$ following the contractions of f in the second stage. This essentially evaluates a composed polynomial that we denote by g at the same point z_i. Since the $\deg g \leq 8$, the composed polynomial can be interpolated by 9 successful evaluations. Hence, it remains to check that $f(\mathcal{T})$ can indeed be recovered from its coefficients.

The master decode $\sum_{i\in[2]}\sum_{-} A_{i,-}B_{i,-}C_{1,-}D_{1,-}E_{1,-}$ by taking a linear combination of $g(2)$ and $g(6)$. Specifically, from a direct computation, we get exactly that the needed quantity equals $[2 \cdot g(2) + 2 \cdot g(6)]/12$. Similarly, the master decode $\sum_{i\in[2]} A_{i,-}B_{i,-}C_{2,-}D_{2,-}E_{2,-}$ by taking a linear combination of $g(\beta_{2,1})$ and $g(\beta_{2,2})$.

B. General Description

For the general scheme, we consider an arbitrary multilinear gradient function f characterized by n gradient-type functions $G_1, \ldots, G_n$, with their degrees denoted by $m_1, m_2, \ldots, m_n \geq 2$. We show that there is a computing scheme, independent of f, that achieves the recovery threshold stated in Theorem 4. Without loss of generality, we assume that $m_1 \leq m_2 \leq \cdots \leq m_n$.

Recall that the dataset is denoted by $X = (X^{\{1\}}, \ldots, X^{\{n\}})$. We say an encoding of X is a *Generalized Lagrange code* if each worker j receives the evaluation of the following polynomial at a distinct point $z_j \in \mathbb{F}$.

$$\tilde{X}_j^{\{k\}} := u^{\{k\}}(z_j) = \sum_{i\in[L_k]} X^{\{k\}(i)} \cdot \Lambda_i^{(k)}(z_j),$$

where $\Lambda_i^{(k)}(z) \triangleq w_i^{(k)} \prod_{\ell\in[L_k]\setminus\{i\}} \prod_{j\in[S_k]} \left(z - \beta_{\ell,j}^{\{k\}}\right)$ is the generalized Lagrange polynomial with coefficients $w_1^{(k)}, \ldots, w_{L_k}^{(k)} \in \mathbb{F}$ and some parameters $\beta_{\ell,j}^{\{k\}}$ and S_k.

To illustrate the main idea in the choice of parameters, we first focus on an elementary case where $m_k = 2$ for all $k \in [n-1]$, while m_n can take an arbitrary integer greater than or equal to 2. In this case, we set $S_k = \prod_{j\in[k-1]} L_j$. Recall that for $i \in [N]$, the i-th worker receives $\tilde{X}_i = (\tilde{X}_i^{\{1\}}, \ldots, \tilde{X}_i^{\{n\}})$ and computes

$$\Gamma\left(g_1\left(\tilde{X}_i^{\{1\}}\right), \ldots, g_n\left(\tilde{X}_i^{\{n\}}\right)\right).$$

Therefore, they essentially evaluate a composed polynomial at point z_i, and we denote that polynomial by g. Hence, by counting the polynomial degree, we have that the coefficient of g can be recovered by a recovery threshold of $(2 + m_n(L_n - 1)) \cdot \prod_{k\in[k-1]} L_k - 1$, which agrees with the expression provided in Theorem 4. Hence, it remains to show that $f(X)$ can indeed be decoded from the coefficient of g by carefully choosing the β and w parameters.

When $n = 1$, by choosing $w_i^{(1)} = \prod_{\ell\in[L_1]\setminus\{i\}} \frac{1}{\beta_{i,1}^{\{1\}} - \beta_{\ell,1}^{\{1\}}}$ and $\beta_{\ell,1}^{\{1\}}$ being distinct elements from $\mathbb{F}$, we have recovered the LCC encoding. Therefore, the master can decode $\Gamma\left(g_1\left(X^{\{1\}(i)}\right)\right)$ by evaluating $g(\beta_{i,1}^{\{1\}})$, and recover $f(X)$ by taking the summation with the linearity of Γ.

For general $n > 1$, we choose $w_i^{(1)}$ and $\beta_{\ell,j}^{\{1\}}$ the same way as the base case. Specifically, note that $S_1 = 1$, $\beta_{\ell,j}^{\{1\}}$ is also specified as any L_1 distinct elements from $\mathbb{F}$. To choose the rest of the parameters, we provide the following convenient lemma. The proof can be found in [45].

Lemma 6. *Given any monic polynomial f with $\deg f = k$ with k distinct roots over a sufficiently large field $\mathbb{F}$, for any polynomial g with $\deg g < k$, there exist sufficiently many $c \in \mathbb{F}$ such that $f + c \cdot g$ has k distinct roots.*

Given this lemma, we define a sequence of parameters recursively. We let $\theta_\ell^{\{1\}} = \beta_{\ell,1}^{\{1\}}$, $p_1(z) := \prod_{\ell\in[L_1]}(z - \beta_{\ell,1}^{\{1\}})$, and $t_1(z) := \sum_{j\in[L_1]} \left(w_j^{(1)}\right)^2 \prod_{\ell\in[L_1]\setminus\{j\}}(z - \beta_{\ell,1}^{\{1\}})$. Then for any $k > 1$, we let $\theta_\ell^{\{k\}}$ be L_k distinct values of c such that $p_{k-1} - ct_{k-1}$ has S_k distinct roots. We let $p_k := \prod_{\ell\in[L_k]}(p_{k-1} - \theta_\ell^{\{k\}} t_{k-1})$, $t_k := t_{k-1} \sum_{j\in[L_k]} \left(w_j^{(k)}\right)^2 \prod_{\ell\in[L_k]\setminus\{j\}}(p_{k-1} - \theta_\ell^{\{k\}} t_{k-1})$, and $w_j^{(k)}$ be arbitrary non-zero elements of $\mathbb{F}$. Recursively, one can prove that t_k and p_k have no common roots, and thus collectively $p_k(z)$ has $L_k S_k$ distinct roots. Hence, we can let the subset of distinct roots of $p_{k-1} - \theta_\ell^{\{k\}} t_{k-1} = 0$ be assigned to the collection of $\{\beta_{\ell,j}^{\{k\}}\}_{j\in[S_k]}$ so that the set of all possible values of β parameters are distinct for each k as well.

Given the above choice of parameters, similar to the base case, after collecting sufficient computational results such that the composed polynomial g is recovered, the master can first recover $\Gamma(G_1(X^{\{1\}}), G_2(X^{\{2\}}), ..., g_n(X^{\{n\}(\ell)}))$ by linearly combining the evaluations of g at $\{\beta_{\ell,j}^{\{n\}}\}_{j\in[S_k]}$, with a coefficient based on the ratio of $\prod_{k\in[S_n]\setminus\{j\}} \frac{1}{\beta_{\ell,j}^{\{n\}} - \beta_{\ell,k}^{\{n\}}}$ and $t_{n-1}(\beta_{\ell,j}^{\{n\}})$. Then, $f(X)$ can be recovered by taking the summation due to the multilinearity of Γ. For detailed proof, see [45].

Finally, to complete the proof for general m, we note that the tensor product of two gradient-type functions can be written as a single gradient-type function, where the L parameter of the new function is the multiplicative product of the original L parameters, while the input consists of all possible pairwise concatenations of the fragments of the original inputs. Therefore, one can merge all gradient functions in f with degree greater than 2, and the scheme we presented for the base case applied to this adapted function achieves the stated recovery threshold.

V. Conclusion and Future Works

This work generalizes the Lagrange coding designs for straggler-resilient multilinear gradient computation. Our proposed design provides a direct while exponential improvement on the recovery threshold for coded tensor network contraction compared to prior works. While we focus on the computation of unweighted gradient functions, our code can be extended to weighted gradient sums to achieve the same performance. Establishing further extension is an interesting future direction.

REFERENCES

[1] Q. Yu, *Coded computing: A transformative framework for resilient, secure, private, and communication efficient large scale distributed computing*. PhD thesis, University of Southern California, 2020.

[2] Q. Yu and A. S. Avestimehr, "Harmonic coding: An optimal linear code for privacy-preserving gradient-type computation," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1102–1106, IEEE, 2019.

[3] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30*, pp. 4406–4416, Curran Associates, Inc., 2017, *arXiv:1705.10464*, 2017.

[4] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *2013 IEEE symposium on security and privacy*, pp. 334–348, IEEE, 2013.

[5] V. Chen, V. Pastro, and M. Raykova, "Secure computation for machine learning with spdz," *arXiv preprint arXiv:1901.00329*, 2019.

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

[7] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltanolkotabi, and S. A. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security, and privacy," in *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1215–1225, PMLR, 2019.

[8] S. Li, S. M. M. Kalan, Q. Yu, M. Soltanolkotabi, and A. S. Avestimehr, "Polynomially coded regression: Optimal straggler mitigation via data encoding," *arXiv preprint arXiv:1805.09934*, 2018.

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

[10] S. Li, S. Avestimehr, *et al.*, "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. 1–148, 2020.

[11] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded fourier transform," in *2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 494–501, IEEE, 2017.

[12] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," *Advances In Neural Information Processing Systems*, vol. 29, 2016.

[13] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrxix multiplication: Fundamental limits and optimal coding," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 2022–2026, June 2018, *arXiv:1801.07487v1*, 2018.

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

[15] F. Pan and P. Zhang, "Simulation of quantum circuits using the big-batch tensor network method," *Physical Review Letters*, vol. 128, no. 3, p. 030501, 2022.

[16] I. L. Markov and Y. Shi, "Simulating quantum computation by contracting tensor networks," *SIAM Journal on Computing*, vol. 38, no. 3, pp. 963–981, 2008.

[17] P. Seitz, I. Medina, E. Cruz, Q. Huang, and C. B. Mendl, "Simulating quantum circuits using tree tensor networks," *Quantum*, vol. 7, p. 964, 2023.

[18] J. Lee, Z. Zhang, S. González-García, and H. Jeong, "Coded computing meets quantum circuit simulation: Coded parallel tensor network contraction algorithm," in *2024 IEEE International Symposium on Information Theory (ISIT)*, pp. 3095–3100, IEEE, 2024.

[19] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, and H. Neven, "Simulation of low-depth quantum circuits as complex undirected graphical models," *arXiv preprint arXiv:1712.05384*, 2017.

[20] S. Aaronson and L. Chen, "Complexity-theoretic foundations of quantum supremacy experiments," *arXiv preprint arXiv:1612.05903*, 2016.

[21] S. Efthymiou, J. Hidary, and S. Leichenauer, "Tensornetwork for machine learning," *arXiv preprint arXiv:1906.06329*, 2019.

[22] Y. Ji, Q. Wang, X. Li, and J. Liu, "A survey on tensor techniques and applications in machine learning," *IEEE Access*, vol. 7, pp. 162950–162990, 2019.

[23] J. Huang, D. A. Matthews, and R. A. van de Geijn, "Strassen's algorithm for tensor contraction," *SIAM Journal on Scientific Computing*, vol. 40, no. 3, pp. C305–C326, 2018.

[24] S. Rajbhandari, A. Nikam, P.-W. Lai, K. Stock, S. Krishnamoorthy, and P. Sadayappan, "A communication-optimal framework for contracting distributed tensors," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 375–386, IEEE, 2014.

[25] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[26] S. Wang, J. Liu, and N. Shroff, "Coded sparse matrix multiplication," in *ICML*, pp. 5152–5160, PMLR, 2018.

[27] M. Soleymani, H. Mahdavifar, and A. S. Avestimehr, "Analog lagrange coded computing," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 283–295, 2021.

[28] A. Asheralieva and D. Niyato, "Fast and secure computational offloading with lagrange coded mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 4924–4942, 2021.

[29] A. Asheralieva, D. Niyato, and Z. Xiong, "Auction-and-learning based lagrange coded computing model for privacy-preserving, secure, and resilient mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 2, pp. 744–764, 2021.

[30] N. Raviv and D. A. Karpuk, "Private polynomial computation from lagrange encoding," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 553–563, 2019.

[31] A. Asheralieva and D. Niyato, "Throughput-efficient lagrange coded private blockchain for secured iot systems," *IEEE Internet of Things Journal*, vol. 8, no. 19, pp. 14874–14895, 2021.

[32] J. Zhu, Q. Yan, X. Tang, and S. Li, "Symmetric private polynomial computation from lagrange encoding," *IEEE Transactions on Information Theory*, vol. 68, no. 4, pp. 2704–2718, 2022.

[33] M. Zinkevich, M. Weimer, L. Li, and A. Smola, "Parallelized stochastic gradient descent," *Advances in neural information processing systems*, vol. 23, 2010.

[34] B. F. Smith, "Domain decomposition methods for partial differential equations," in *Parallel numerical algorithms*, pp. 225–243, Springer, 1997.

[35] P. (california State University Renteln (San Bernardino)), *Manifolds, Tensors, and Forms-an Introduction for Mathematicians and Phys*. Cambridge University Press, 2013.

[36] G. Evenbly, "A practical guide to the numerical implementation of tensor networks i: Contractions, decompositions, and gauge freedom," *Frontiers in Applied Mathematics and Statistics*, vol. 8, p. 806549, 2022.

[37] S. Montangero, E. Montangero, and Evenson, *Introduction to tensor network methods*. Springer, 2018.

[38] R. Orús, "Tensor networks for complex quantum systems," *Nature Reviews Physics*, vol. 1, no. 9, pp. 538–550, 2019.

[39] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[40] E. Robeva and A. Seigal, "Duality of graphical models and tensor networks," *Information and Inference: A Journal of the IMA*, vol. 8, pp. 273–288, 06 2018.

[41] C. Huang, F. Zhang, M. Newman, X. Ni, D. Ding, J. Cai, X. Gao, T. Wang, F. Wu, G. Zhang, *et al.*, "Efficient parallelization of tensor network contraction for simulating quantum computation," *Nature Computational Science*, vol. 1, no. 9, pp. 578–587, 2021.

[42] T. Vincent, L. J. O'Riordan, M. Andrenkov, J. Brown, N. Killoran, H. Qi, and I. Dhand, "Jet: Fast quantum circuit simulations with parallel task-based tensor-network contraction," *Quantum*, vol. 6, p. 709, 2022.

[43] J. Chen, F. Zhang, C. Huang, M. Newman, and Y. Shi, "Classical simulation of intermediate-size quantum circuits," *arXiv preprint arXiv:1805.01450*, 2018.

[44] Q. Yu and A. S. Avestimehr, "Coded computing for resilient, secure, and privacy-preserving distributed matrix multiplication," *IEEE Transactions on Communications*, vol. 69, no. 1, pp. 59–72, 2021.

[45] K. Wang, Z. Zhang, and Q. Yu, "Lagrange coding for tensor network contraction: Achieving polynomial recovery thresholds," 2025. online preprint.