Coded Computing
A Transformative Framework for Resilient, Secure, and Private Distributed Learning

Salman Avestimehr (USC)

In collaboration with: Qian Yu, Songze Li, Basak Guler, Jinyun So, Payman Mohassel, and Mahdi Soltanolkotabi
LIDS Seminar, Feb, 2019
Motivating Example

- Consider a hospital that wants to train a machine learning model using patients’ healthcare records
Training a machine learning model is a compute and storage intensive task that is desired to be offloaded to cloud/edge.
Motivating Example

- Training a machine learning model is a compute and storage intensive task that is desired to be offloaded to cloud/edge.

How to develop a resilient, secure, and privacy-preserving framework for distributed computing/learning?
Setting

Wants to compute $f(X_1)$, $f(X_2)$, ..., $f(X_K)$
Wants to compute $f(X_1), f(X_2), \ldots, f(X_K)$

- What if some nodes straggle/fail?
- What if some nodes are malicious?
- What if we want to keep data private?
Wants to compute \( f(X_1), f(X_2), \ldots, f(X_K) \)

e.g., to tolerate one straggler each computation is repeated twice!
e.g., to tolerate one error each computation is repeated three times!

Setting

- What if some nodes straggle/fail?
- What if some nodes are malicious?
- What if we want to keep data private?

inefficient replication
differential privacy
homomorphic encryption
secure multi-party computing

Wants to compute \( f(X_1), f(X_2), \ldots, f(X_K) \)
Our proposal: **Coded Computing**

- Shannon’s Coding Theory
- von Neumann’s Computing Theory

Coded Computing
Our proposal: **Coded Computing**

- Shannon’s Coding Theory
- von Neumann’s Computing Theory

Communication channel
Our proposal: Coded Computing

Shannon’s Coding Theory

von Neumann’s Computing Theory
Our proposal: **Coded Computing**

Shannon’s Coding Theory

von Neumann’s Computing Theory

[Diagram of a computer system with labeled units: Central Processing Unit, Control Unit, Arithmetic/Logic Unit, Memory Unit, Input Device, Output Device]
Our proposal: **Coded Computing**

Shannon’s Coding Theory

von Neumann’s Computing Theory
Coded Computing in a Nutshell

inefficient replication!
Coded Computing in a Nutshell

Key Challenge: how to design codes so that computation on coded data is meaningful?

special case of “linear functions” have been the focus of most (all) prior works, MDS coding, ShortDot, sparse coding, ...

Note that for linear functions coding and computation commute: \( f(x_1+x_2) = f(x_1) + f(x_2) \)

Key idea: injecting computation redundancy in unorthodox coded forms.
Bi-Linear Computation: Massive Matrix Multiplication

The key algebraic building block of many ML algorithms
Bi-Linear Computation: Massive Matrix Multiplication
Bi-Linear Computation: Massive Matrix Multiplication

Slowest server determines the speed of computation!

How to deal with stragglers/failures?
Naïve Repetition

8 servers used, 1 straggler is tolerated
Can we tolerate 4 stragglers (the same as MDS codes)??!!
Polynomial Codes [NIPS 2017]

\[
\begin{bmatrix}
A_0^T \\
A_1^T
\end{bmatrix}
\begin{bmatrix}
B_0, B_1
\end{bmatrix}
\]

\[f(x) = A_0 + A_1x\]

\[g(x) = B_0 + B_1x^2\]

Server \(i\) calculates: \(h(i) = f^T(i)g(i)\)
Polynomial Codes [NIPS 2017]

\[ f(x) = A_0 + A_1 x \]
\[ g(x) = B_0 + B_1 x^2 \]
Polynomial Codes [NIPS 2017]

\[
\begin{bmatrix}
A_0^T \\
A_1^T
\end{bmatrix}
\begin{bmatrix}
B_0, B_1
\end{bmatrix}
\]

\[f(x) = A_0 + A_1 x\]
\[g(x) = B_0 + B_1 x^2\]

\[h(x) = f^T(x)g(x) = (A_0^T + A_1^T x)(B_0 + B_1 x^2)\]

\[h(x) = f^T(x)g(x) = A_0^T B_0 + A_1^T B_0 x + A_0^T B_1 x^2 + A_1^T B_1 x^3\]

Result of any 4 servers is enough to calculate \(A^T B\).

The code is Reed-Solomon after multiplication. It is MDS!
Coded Computing for more General Computations?

Wants to compute \( f(X_1), f(X_2), \ldots, f(X_k) \)

- arbitrary multivariate polynomial \( f \)
  - Matrix algebra
  - Tensor algebra
  - Typical data processing (filtering, ...)
  - General loss functions in ML algorithms

Wants to compute \( f(X_1), f(X_2), \ldots, f(X_k) \)
Lagrange Coded Computing [AISTATS 2019]

• Data encoding
  – Pick distinct $\beta_1, \beta_2, \ldots, \beta_K$
  – Construct Lagrange polynomial $u(z) \triangleq \sum_{j=1}^{K} X_j \cdot \prod_{k \neq j} \frac{z - \beta_k}{\beta_j - \beta_k}$
  – Pick distinct $\alpha_1, \alpha_2, \ldots, \alpha_N$
  – Compute $\tilde{X}_i = u(\alpha_i)$
Local computing

- Worker $i$ computes $f(\tilde{X}_i) = f(u(\alpha_i))$
- This is equivalent to evaluate the polynomial $f(u(z))$ of degree $(K - 1) \deg f$ at $\alpha_i$
Lagrange Coded Computing

- Computation decoding (recovering $f(X_1), \ldots, f(X_K)$)
  - Master interpolates $f(u(z))$ after receiving results from any $(K - 1) \text{deg } f + 1$ workers
  - Evaluate at $z = \beta_j$ to recover $f(u(\beta_j)) = f(X_j)$
Lagrange Coded Computing

- **Data encoding**
  - Pick distinct $\beta_1, \beta_2, \ldots, \beta_K$
  - Construct Lagrange polynomial $u(z) \triangleq \sum_{j=1}^{K} X_j \prod_{k \neq j} \frac{z - \beta_k}{\beta_j - \beta_k}$
  - Pick distinct $\alpha_1, \alpha_2, \ldots, \alpha_N$
  - Compute $\tilde{X}_i = u(\alpha_i)$

- **Local computing**
  - Worker $i$ computes $f(\tilde{X}_i) = f(u(\alpha_i))$
  - This is equivalent to evaluate the polynomial $f(u(z))$ of degree $(K - 1)$ deg $f$ at $\alpha_i$

- **Computation decoding**
  - Master interpolates $f(u(z))$ after receiving results from any $(K - 1) \text{deg } f + 1$ workers
  - Evaluate at $z = \beta_j$ to recover $f(u(\beta_j)) = f(X_j)$
Lagrange Coded Computing

**Theorem:**

To evaluate an arbitrary multivariate polynomial $f$ on $K$ input data blocks using $N$ workers, the optimal recovery threshold $T^*$ is

$$T^* = (K - 1) \deg f + 1$$

- Applies to arbitrary polynomials beyond linear functions (General Matrix algebra, tensor algebra, loss functions in ML, …)
- A replication scheme would need the results of $\frac{K - 1}{K} N + 1$
  - Example (N=100, K=10, deg=2): LCC needs the results of 19 workers while replication schemes need 91!
- The optimal recovery threshold of LCC does not scale with $N$
  - Adding one more worker, increases the resiliency of LCC by 1
  - Faster computation using more workers
- Lagrange Coded Computing (LCC) maps edge computing to polynomial interpolation that can be solved effectively using information and coding theories

How about malicious nodes?

**Theorem:**

To evaluate an arbitrary multivariate polynomial $f$ on $K$ input data blocks using $N$ workers, with possibly $A$ adversary nodes, the optimal recovery threshold $T^*$ is

$$T^* = (K - 1) \deg f + 2A + 1$$

Adding one more worker increases the resiliency to adversaries by $1/2$. 
How about private computing?

- If there are $T$ colluding workers: pad the dataset $(X_1, \ldots, X_K)$ set with $Z_1, \ldots, Z_T$
- We need to only recover a polynomial with higher degree

$$u(z) \triangleq \sum_{j \in [K]} X_j \cdot \prod_{k \in [K+T] \setminus \{j\}} \frac{z - \beta_k}{\beta_j - \beta_k}$$

$$+ \sum_{j = K+1} \prod_{k \in [K+T] \setminus \{j\}} \frac{z - \beta_k}{\beta_j - \beta_k}.$$
Example

- Data set, $X_1$, $X_2$ (square matrices)
- Computation: $f(X) = X^2$
- $N=7$ workers
- Guarantee information-theoretic privacy of the data set at each worker

\[
u(z) = X_1 \cdot \frac{(z-2)(z-3)}{(1-2)(1-3)} + X_2 \cdot \frac{(z-1)(z-3)}{(2-1)(2-3)} + Z \cdot \frac{(z-1)(z-2)}{(3-1)(3-2)},\]

\[
\tilde{X}_1 = u(\alpha_1) \quad \tilde{X}_2 = u(\alpha_2) \quad \tilde{X}_3 = u(\alpha_3) \quad \tilde{X}_4 = u(\alpha_4) \quad \tilde{X}_5 = u(\alpha_5) \quad \tilde{X}_6 = u(\alpha_6) \quad \tilde{X}_7 = u(\alpha_7)
\]

\[
f(\tilde{X}_1) \quad f(\tilde{X}_2) \quad f(\tilde{X}_3) \quad f(\tilde{X}_4) \quad f(\tilde{X}_5) \quad f(\tilde{X}_6) \quad f(\tilde{X}_7)
\]

$f(u(z))$ is a degree 4 polynomial can also tolerate 1 adversary
How about private computing?

**Theorem:**

To evaluate an arbitrary multivariate polynomial $f$ on $K$ input data blocks using $N$ workers, with possibly $A$ adversary $T$ colluding nodes, Lagrange Coded Computing achieves recovery threshold of

$$(K + T - 1) \deg f + 2A + 1$$
Application to Distributed Learning

- The loss calculation in gradient methods can be modeled as a polynomial
- Can leverage LCC to speed-up computations

Polynomially Coded Regression: Optimal Straggler Mitigation via Data Encoding

Songze Li, Seyed Mohammadreza Mousavi Kalan, Qian Yu, Mahdi Soltanolkotabi, and A. Salman Avestimehr
Department of Electrical Engineering, University of Southern California, Los Angeles, CA, USA

- Linear regression
- 40 workers
- Amazon EC2

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Uncoded</th>
<th>GC</th>
<th>PCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>1.857 s</td>
<td>1.373 s</td>
<td>1.331 s</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>1.879 s</td>
<td>1.373 s</td>
<td>1.331 s</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>2.401 s</td>
<td>2.019 s</td>
<td>1.674 s</td>
</tr>
<tr>
<td>Scenario 4</td>
<td>2.216 s</td>
<td>1.857 s</td>
<td>1.407 s</td>
</tr>
</tbody>
</table>

We plot the CDFs of the per iteration run-time for the PCR and BCC schemes in the four scenarios in Figure 4.
Application to Blockchains

- There is a surge of interest to use “sharding” to increase the efficiency and throughput of blockchains, but …
Application to Blockchains

- We have developed the concept of **coded sharding**, in particular **polyshard** that leverages LCC
Application to Blockchains

- We have developed the concept of coded sharding, in particular polyshard that leverages LCC

PolyShard: Coded Sharding Achieves Linearly Scaling Efficiency and Security Simultaneously

Songze Li*, Mingchao Yu*, A. Salman Avestimehr*, Sreeram Kannan†, and Pramod Viswanath‡

<table>
<thead>
<tr>
<th></th>
<th>Storage efficiency</th>
<th>Security</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full replication</td>
<td>$O(1)$</td>
<td>$\Theta(N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Sharding</td>
<td>$\Theta(N)$</td>
<td>$O(1)$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>Information-theoretic limit</td>
<td>$\Theta(N)$</td>
<td>$\Theta(N)$</td>
<td>$\Theta(N)$</td>
</tr>
<tr>
<td>PolyShard (this paper)</td>
<td>$\Theta(N)$</td>
<td>$\Theta(N)$</td>
<td>$\Theta(N)$</td>
</tr>
</tbody>
</table>

* University of Southern California
† University of Washington
‡ University of Toronto
Application to Private Machine Learning: CodedPrivateML
Setting

How to offload the training task to a distributed platform while keeping the dataset and model private?

- Train a logistic (or linear) regression model
- Any collusions between up to $T$ from $N$ workers reveals no information about training data and the model

worker 1 \ldots worker N

T colluding workers
Potential Approach 1

**Differential Privacy:** Mainly used when model is to be released to public

- **Drawbacks:**
  - *Trades accuracy with privacy:* Stronger privacy requires more noise
  - Doesn’t provide **strong** privacy
  - Protects only the privacy of personally identifiable information (removal of one data point does not change the model significantly)

[Chaudhuri-Monteleoni 09, Abadi et al. 16, McMahan et al. 18, Rajkumar-Agarwal 12, Jayaraman et al. 18]
Potential Approach 2

Homomorphic Encryption

- Privacy based on computational assumptions (as opposed to information-theoretic)
- Computations in encrypted domain
- Orders of magnitude slowdown (training on MNIST data takes ~2 hours)

[Gilad-Bachrach et al. 16; Hesamifard et al. 17, Graepel et al. 12, Kim et al. 18, Wang et al. 18, Han et al. 19]
Potential Approach 3

Secure Multi-Party Computing (MPC)

Drawbacks:
- Extensive communication between parties that limits scalability
- No benefits from parallelization
- Computation load at each party is as high as training centrally

[Mohassel-Zhang 17, Nikolaenko et al. 13, Gascon et al. 17, Dahl et al. 18, Chen et al. 19, Mohassel-Rindal]
Master offloads computationally-intensive operations (gradient computations) to N workers
Problem Setting

Information-theoretic privacy guarantee for both the data and the model.

Dataset: $X = (X_1, \ldots, X_K)$

$$I(X; \bar{X}_T, \{\bar{W}_T^{(t)}\}_{t \in [J]}) = 0, \forall T \subset [N], |T| \leq T.$$
Overview of CodedPrivateML

\[ x = (x_1, \ldots, x_K) \]
\[ y \]

\[ \tilde{x}_1, \tilde{W}_1^{(t)} \]
\[ \tilde{x}_N, \tilde{W}_N^{(t)} \]

Secret sharing of the dataset

Secret sharing of model parameters

Local computation at the workers

Decoding and model update

Until convergence

\[ \tilde{W}^{(t+1)} \]
Secret Sharing of the Dataset and Model Parameters

Key properties.
• Privacy preserving
• Enabling fast and accurate computations on secret shares

Step 1: Quantization.
Use quantization to convert between real & finite domains

• **Quantized dataset:** $\overline{X}$ (deterministic rounding of $X$)
• **Quantized weights:** $\overline{W}^{(t)} = [\overline{w}^{(t),1}, \ldots, \overline{w}^{(t),r}]$

independent stochastic quantizations of $w^{(t)}$
Secret Sharing of the Dataset and Model Parameters

**Step 2: Lagrange encoding.**

Use Lagrange coding for secret sharing the dataset and model parameters.

Lagrange interpolation polynomial:

\[
\begin{align*}
    u(z) &= \sum_{j \in [K]} \overline{X_j} \cdot \prod_{k \in [K+T] \setminus \{j\}} \frac{z - \beta_k}{\beta_j - \beta_k} \\
    &\quad + \sum_{j=K+1}^{K+T} Z_j \cdot \prod_{k \in [K+T] \setminus \{j\}} \frac{z - \beta_k}{\beta_j - \beta_k}.
\end{align*}
\]
Comparison with MPC

- Secure/Private multi-party computing (MPC) also aims at solving the same problem
- Shamir’s secret sharing scheme is commonly used for private data sharing (e.g., BGW scheme)

\[ P_i(z) = X_i + Z_{i,1}z + \ldots + Z_{i,T}z^T \]
Table 1: Comparison between BGW based designs and LCC

<table>
<thead>
<tr>
<th>Complexity per worker</th>
<th>BGW</th>
<th>LCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frac. data per worker</td>
<td>$K$</td>
<td>$1$</td>
</tr>
<tr>
<td>Randomness</td>
<td>$KT$</td>
<td>$T$</td>
</tr>
</tbody>
</table>

In the BGW scheme, the computation is distributed via LCC. In other words, data encoding of LCC can be done for any polynomial up to a certain degree, as previously mentioned. A BGW scheme was also proposed in [5]. The computation is then carried out as follows:

Given inputs in secure and private computing, let us consider the coded computing literature. This work extends the BGW first uses Shamir’s scheme for every $\tilde{X}_i = u(\alpha_i)$. After computation, each worker has essentially evaluated the polynomials $f$ at the points $\tilde{Z}_i, \ldots, \tilde{Z}_T$. For example, setting $\alpha_1 = \cdots = \alpha_T = 1$, this gives rise to the second key advantage of LCC recovers several previously studied results as special cases. For example, setting $\alpha_1 = \cdots = \alpha_T = 1$, this gives rise to the second key advantage of LCC.

Contrary to the BGW scheme, which is larger than that of the BGW scheme, the BGW scheme will then store $\frac{K}{2}$ of the data. In the LCC scheme, on the other hand, each worker needs to store $\frac{1}{K}$ of the data. The amount of randomness in LCC is significantly reduced by evaluating $(\tilde{Z}_1, \ldots, \tilde{Z}_T)$’s, in which the master can recover all required results by evaluating $f(\tilde{Z}_1, \ldots, \tilde{Z}_T)$. This results in significant reduction in the amount of randomness. Note that under the same condition, LCC scheme requires more workers to guarantee the security and privacy.
Secret Sharing of the Dataset and Model Parameters

**Step 2: Lagrange encoding.**

Compute \( f(\tilde{X}_1, \tilde{W}_1^{(t)}), \ldots, f(\tilde{X}_N, \tilde{W}_N^{(t)}) \)

**Challenge:** Lagrange encoding is designed for polynomial computations, but logistic regression includes non-polynomial computations due to the sigmoid function.
Secret Sharing of the Dataset and Model Parameters

**Step 3: Polynomial Approximation.**

- Approximate the sigmoid with a polynomial function.

\[
\hat{g}(z) = \sum_{i=0}^{r} c_i z^i
\]
Local Computation at the Workers

Worker \( i \in [N] \) locally computes

\[
f(\tilde{X}_i, \tilde{W}^{(t)}_i) = \tilde{X}_i^T \hat{g}(\tilde{X}_i, \tilde{W}^{(t)}_i)
\]

and sends the result to master.
After receiving the results from a sufficient number of workers, master:

- decodes the local gradients using polynomial interpolation
- aggregates the local gradients
- converts the result to real domain
- updates the model
System Overview of CodedPrivateML

\[
X = (x_1, \ldots, x_K)
\]

\[
y
\]

\[
\bar{x}_1, \bar{W}_1^{(t)}
\]

\[
\bar{x}_N, \bar{W}_N^{(t)}
\]

\[
\text{master}
\]

\[
\text{worker 1}
\]

\[
\text{worker N}
\]

Secret sharing of the dataset

Secret sharing of model parameters

Local computation at the workers

Decoding and model update

\[
W^{(t+1)}
\]

Until convergence
Convergence and Privacy Guarantees

**Theorem.** For the distributed training of a logistic regression model with $N$ workers, given a dataset $X = (X_1, \ldots, X_K)$, CodedPrivateML guarantees:

(Convergence) $w^{(t)}$ converges to $w^{(*)}$,

(Privacy) $X$ remains information-theoretically private against any $T$ colluding workers,

$$I(X; \tilde{X}_T, \{\tilde{W}^{(t)}_T\}_{t \in [J]}) = 0, \forall T \subset [N], |T| \leq T,$$

as long as $N \geq (2r + 1)(K + T - 1) + 1$. 

Privacy–Parallelization Trade-off

Given N workers, CodedPrivateML can achieve any K and T, as long as:

\[ N \geq (2r + 1)(K + T - 1) + 1 \]

**Parallelization increases with K:**
Computation load at each worker is proportional to 1/K-th of processing the entire dataset.

**Privacy increases with T:**
Dataset remains private against T colluding workers.
Experiments

- Implementation on Amazon EC2 Cloud
- Binary image classification on the MNIST dataset
- CodedPrivateML vs. secure MPC applied to our problem
- MPC-based scheme: BGW protocol [Ben-Or et al. ’88]
  - similar privacy structure (T out of N)
  - information-theoretic privacy
Experiments

Training for 95.04% accuracy (25 iterations)

Case 1 (maximum parallelization):
All resources to parallelization
\[ K = \left\lfloor \frac{(N - 1)}{3} \right\rfloor, \quad T = 1 \]

Case 2 (equal parallelization and privacy): Resources split equally
\[ K = T = \left\lfloor \frac{(N + 2)}{6} \right\rfloor \]
Experiments

Accuracy

Convergence

**CodedPrivateML:** 95.04%

**Conventional logistic regression:** 95.98%
Conclusion

• Coded computing is a new promising approach to alleviate key bottlenecks in distributed computing/learning
  – Latency
  – Resiliency
  – Security and privacy
  – Bandwidth consumption (see CodedMapReduce)

• Coded Computing can be applied to various applications in distributed learning, fault tolerant computing, blockchains, …

• There are many exciting research problems ahead
  – Generalization to broader class of computations (significant progress made in Lagrange coding)
  – Multi-stage and iterative computations in ML
  – Heterogeneous and asymmetric computations
  – …