ApproxIFER: A Model-Agnostic Approach to Resilient and Robust Prediction Serving Systems

Mahdi Soleymani[1], **Ramy E. Ali**[2], **Hessam Mahdavifar**[1], **A. Salman Avestimehr**[2]
[1] University of Michigan - Ann Arbor
[2] University of Southern California (USC)
mahdy@umich.edu, reali@usc.edu, hessam@umich.edu, avestime@usc.edu

Abstract

Due to the surge of cloud-assisted AI services, the problem of designing resilient prediction serving systems that can effectively cope with stragglers and minimize response delays has attracted much interest. The common approach for tackling this problem is replication which assigns the same prediction task to multiple workers. This approach, however, is inefficient and incurs significant resource overheads. Hence, a learning-based approach known as parity model (ParM) has been recently proposed which learns models that can generate "parities" for a group of predictions to reconstruct the predictions of the slow/failed workers. While this learning-based approach is more resource-efficient than replication, it is tailored to the specific model hosted by the cloud and is particularly suitable for a small number of queries (typically less than four) and tolerating very few stragglers (mostly one). Moreover, ParM does not handle Byzantine adversarial workers. We propose a different approach, named Approximate Coded Inference (ApproxIFER), that does not require training any parity models, hence it is agnostic to the model hosted by the cloud and can be readily applied to different data domains and model architectures. Compared with earlier works, ApproxIFER can handle a general number of stragglers and scales significantly better with the number of queries. Furthermore, ApproxIFER is robust against Byzantine workers. Our extensive experiments on a large number of datasets and model architectures show significant degraded mode accuracy improvement by up to 58% over ParM.

Introduction

Machine learning as a service (MLaS) paradigms allow incapable clients to outsource their computationally-demanding tasks such as neural network inference tasks to powerful clouds (Amazon 2021; Microsoft 2021; Google 2021). More specifically, prediction serving systems host complex machine learning models and respond to the inference queries by the corresponding predictions with low latency. To ensure a fast response to the different queries in the presence of stragglers, prediction serving systems distribute these queries on multiple workers in the system each having an instance of the deployed model (Crankshaw et al. 2017). Such systems mitigate stragglers through replication which assigns the same task to multiple workers, either proactively or reactively, in order to reduce the tail latency of the computations (Suresh et al. 2015; Dean and Barroso 2013; Shah, Lee, and Ramchandran 2015; Gardner et al. 2015; Chaubey and Saule 2015). Replication, however, entails significant overhead as a result of assigning the same task to multiple workers.

Erasure coding is known to be more resource-efficient compared to replication and has been recently leveraged to speed up distributed computing and learning systems (Lee et al. 2017; Yu et al. 2019; Yu, Maddah-Ali, and Avestimehr 2017; Narra et al. 2019; Muralee Krishnan, Hosseini, and Khisti 2020; Soto, Li, and Fan 2019). The traditional coding-theoretic approaches, known as the coded computing approaches, are limited to polynomial computations and require a large number of workers that depends on the desired computation. Hence, such techniques cannot be directly applied in prediction serving systems.

To overcome the limitations of the traditional coding-theoretic approaches, a learning-based approach known as ParM was proposed in (Kosaian, Rashmi, and Venkataraman 2019). In ParM, the prediction queries are encoded using an erasure code. These coded queries are then transformed into coded predictions by learning *parity models* to provide straggler resiliency. The desired predictions are then reconstructed from the fastest workers as shown in Fig. 1. By doing this, ParM can be applied to non-polynomial computations with a number of workers that is independent of the computations.

These parity models, however, depend on the model hosted by the cloud and are suitable for tolerating one straggler and handling a small number of queries (typically less than 4). Moreover, they require retraining whenever they are used with a new hosted model. In this work, we take a different approach leveraging approximate coded computing techniques (Jahani-Nezhad and Maddah-ali 2022) to design scalable, straggler-resilient and Byzantine-robust prediction serving systems. Our approach relies on rational interpolation techniques (Berrut 1988) to estimate the predictions of the slow and erroneous workers from the predictions of the other workers. Our contributions in this work are summarized as follows.

1. We propose ApproxIFER, a model-agnostic inference framework that leverages approximate computing techniques. In ApproxIFER, all workers deploy instances of the model hosted by the cloud and no additional models are required as shown in Fig. 2. Furthermore, the encod-

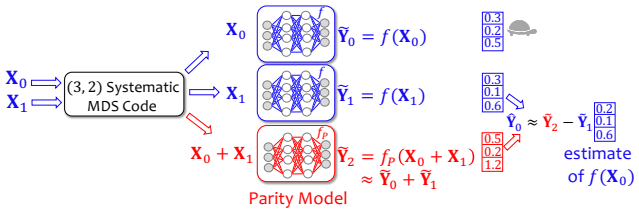

Figure 1: An example of ParM is illustrated with $K = 2$ queries denoted by $\mathbf{X}_0$ and $\mathbf{X}_1$. The goal is to compute the predictions $\mathbf{Y}_0 = f(\mathbf{X}_0)$ and $\mathbf{Y}_1 = f(\mathbf{X}_1)$. In this example, the system is designed to tolerate one straggler. Worker 1 and worker 2 have the model deployed by the prediction serving system denoted by f. Worker 3 has the parity model f_P which is trained with the ideal goal that $f_P(\mathbf{X}_0 + \mathbf{X}_1) = f(\mathbf{X}_0) + f(\mathbf{X}_1)$. In this scenario, the first worker is slow and $f(\mathbf{X}_0)$ is estimated from $f(\mathbf{X}_1)$ and $f_P(\mathbf{X}_0 + \mathbf{X}_1)$.

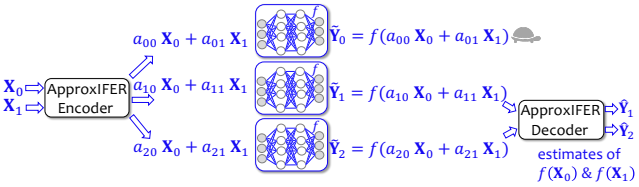

Figure 2: An example of ApproxIFER is illustrated with $K = 2$ queries and $S = 1$ straggler. Unlike ParM, all workers in ApproxIFER have the same model f which is the model hosted by the cloud. In this scenario, the first worker is slow and $f(\mathbf{X}_0)$ and $f(\mathbf{X}_1)$ are estimated from $\tilde{\mathbf{Y}}_1$ and $\tilde{\mathbf{Y}}_2$. The key idea of ApproxIFER is that it carefully chooses the coefficients while encoding the queries such that it can estimate the desired predictions from the coded predictions of the fast workers through interpolation.

ing and the decoding procedures of ApproxIFER do not depend on the hosted model. This enables ApproxIFER to be easily applied to any neural network architecture.

2. ApproxIFER is also robust to erroneous workers that return incorrect predictions either unintentionally or adversarially. To do so, we have proposed an algebraic interpolation algorithm to decode the desired predictions from the erroneous coded predictions.
   ApproxIFER requires a significantly smaller number of workers than the conventional replication method. More specifically, to handle K prediction queries while tolerating up to E Byzantine adversarial workers, ApproxIFER requires only $2K + 2E$ workers whereas the replication-based schemes require $(2E + 1)K$ workers. Moreover, ApproxIFER can be set to tolerate any number of stragglers S and errors E efficiently while scaling well with the number of queries K, whereas the prior works focused on the case where $S = 1, E = 0$ and $K = 2, 3, 4$.
3. We run extensive experiments on the MNIST, Fashion-MNIST, CIFAR-10 and ImageNet datasets on the VGG, ResNet, DenseNet, and GoogLeNet architectures which show that ApproxIFER improves the degraded mode accuracy by up to 58% compared to the prior approaches for large K. The results of one of our experiments on ResNet are shown in Fig. 3, but we later report extensive experiments on those datasets and architectures showing a consistent significant accuracy improvement over the prior works.

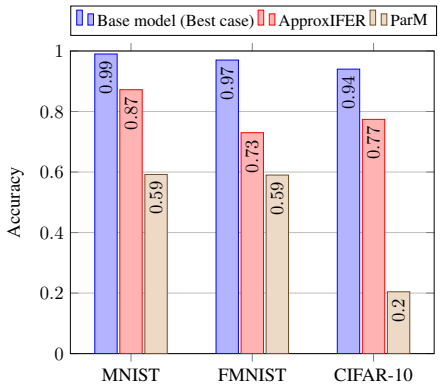

Figure 3: Comparison of the base model test accuracy with the accuracy of ApproxIFER and the degraded mode accuracy of ParM for ResNet-18 and $K = 10$, $S = 1$ and $E = 0$.

Problem Setting

System Architecture. We consider a prediction serving system with N workers. The prediction serving system is hosting a machine learning model denoted by f. We refer to this model as the hosted or the deployed model. Unlike ParM (Kosaian, Rashmi, and Venkataraman 2019), all workers have the same model f in our work as shown in Fig. 4. The input queries are grouped such that each group has K queries. We denote the set of K queries in a group by $\mathbf{X}_0, \mathbf{X}_1, \cdots, \mathbf{X}_{K-1}$.

Goal. The goal is to compute the predictions $\mathbf{Y}_0 = f(\mathbf{X}_0), \mathbf{Y}_1 = f(\mathbf{X}_1), \cdots, \mathbf{Y}_{K-1} = f(\mathbf{X}_{K-1})$ while tolerating up to S stragglers and up to E Byzantine workers.

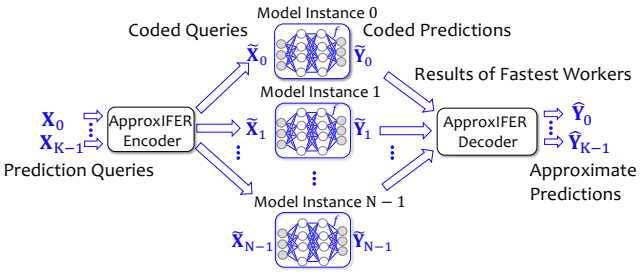

Figure 4: In ApproxIFER, all workers have the model deployed by the system f, no parity models are required and only an encoder and a decoder are added on top of the conventional replication-based prediction serving systems. The K input queries $\mathbf{X}_0, \cdots, \mathbf{X}_{K-1}$ are first encoded. The predictions are then performed on the coded queries. Finally, the approximate predictions $\hat{\mathbf{Y}}_0, \cdots, \hat{\mathbf{Y}}_{K-1}$ are recovered from the fastest workers.

ApproxIFER Algorithm

In this section, we present our proposed protocol based on leveraging approximate coded computing (Jahani-Nezhad and Maddah-ali 2022). The encoder and the decoder of ApproxIFER are based on rational functions and rational interpolation. Most coding-theoretic approaches for designing straggler-resilient and Byzantine-robust distributed systems rely on polynomial encoders and polynomial interpolation for decoding. Polynomial interpolation, however, is known to be unstable (Berrut and Klein 2014). On the other hand, rational interpolation is known to be extremely stable and can lead to faster convergence compared to polynomial interpolation (Berrut 1988). This motivated a recent work to leverage rational functions rather than polynomials to design straggler-resilient distributed training algorithms (Jahani-Nezhad and Maddah-ali 2022). While (Jahani-Nezhad and Maddah-ali 2022) only handles stragglers, ApproxIFER jointly handles both stragglers and Byzantine workers. We provide a very brief background about rational functions next.

Rational Interpolation Background. Consider a function f, n distinct points $a \leq x_0 < x_1, \cdots < x_{n-1} \leq b$ and the corresponding evaluations of f at these points denoted by $f_0, f_1, \cdots, f_{n-1}$. Berrut's rational interpolant of f is then defined as follows (Berrut 1988):

$$r(x) \stackrel{\text{def}}{=} \sum_{i=0}^{n-1} f_i \ell_i(x), \tag{1}$$

where $\ell_i(x)$, for $i \in [n-1]$, where $[n] \stackrel{\text{def}}{=} \{0, 1, 2, \ldots, n\}$, are the basis functions defined as follows

$$\ell_i(x) \stackrel{\text{def}}{=} \frac{(-1)^i}{(x - x_i)} \Big/ \sum_{i=0}^{n-1} \frac{(-1)^i}{(x - x_i)}, \tag{2}$$

for $i \in [n]$. Berrut's rational interpolant has several useful properties as it has no pole on the real line (Berrut 1988) and it is extremely well-conditioned (Bos, De Marchi, and Hormann 2011; Bos et al. 2013).

Rational Interpolation with Erroneous Evaluations. We now provide our proposed error-locator algorithm for rational interpolation in the presence of Byzantine errors. All the details on how this method works along with the theoretical guarantee and proofs are moved to the Appendix A due to space limitations. Let A_{avl} denote the set of indices corresponding to $N - S$ available evaluations of $r(x)$ over x_i's for some S that denotes the number of stragglers. Let A_{adv}, with $|A_{\text{adv}}| \leq E$, denotes the set of indices corresponding to erroneous evaluations. For $i \in A_{\text{avl}}$, let also y_i denote the available and possibly erroneous evaluation of $r(x)$ at x_i. Then we have $y_i = r(x_i)$, for at least $N - S - E$ indices $i \in A_{\text{avl}}$. The proposed algorithm is mainly inspired by the well-known Berlekamp–Welch (BW) decoding algorithm for Reed-Solomon codes in coding theory (Blahut 2008). We tailor the BW algorithm to get a practical algorithm for rational functions that overcomes the numerical issues arising from inevitable round-off errors in the implementation. This algorithm is provided next.

Note that the equations in Step 1 of Algorithm 1 form a homogeneous system of linear equations with $2(K + E)$ unknown variables where the number of equations is $N - S$. In order to guarantee the existence of a non-trivial solution, we must have

$$N \geq 2K + 2E + S. \tag{3}$$

This guarantees the existence of a solution to $P(x)$ and $Q(x)$.

Next, the encoding and decoding algorithms of ApproxIFER are discussed in detail.

ApproxIFER Encoding. The K input queries $\mathbf{X}_j$, for $j \in [K-1]$, are first encoded into N coded queries, denoted by $\tilde{\mathbf{X}}_i$, for $i \in [N-1]$, each given to a worker. As mentioned earlier, the aim is to provide resilience against any S straggler workers and robustness against any E Byzantine adversarial workers. When $E = 0$, we assume that $N = K + S$ which corresponds to an overhead of $\frac{K+S}{K}$. Otherwise, $N = 2(K + E) + S$ which corresponds to an overhead of $\frac{2(K+E)+S}{K}$. In general, the overhead is defined as the number of workers divided by the number of queries.

To encode the queries, we leverage Berrut's rational interpolant discussed as follows (Jahani-Nezhad and Maddah-ali 2022). First, a rational function u is computed in such a way that it passes through the queries. More specifically,

$$u(z) = \sum_{j \in [K-1]} \mathbf{X}_j \ell_j(z), \tag{4}$$

where $\ell_j(x)$, for $j \in [K-1]$, are the basis functions defined as follows

$$\ell_j(z) = \frac{(-1)^j}{(z - \alpha_j)} \Big/ \sum_{j \in [K-1]} \frac{(-1)^j}{(z - \alpha_j)}, \tag{5}$$

and α_j is selected as a Chebyshev point of the first kind as

$$\alpha_j = \cos \frac{(2j+1)\pi}{2K} \tag{6}$$

for all $j \in [K-1]$. The queries are then encoded using this rational function as follows

$$\tilde{\mathbf{X}}_i \stackrel{\text{def}}{=} u(\beta_i), \tag{7}$$

where β_i is selected as a Chebyshev point of the second kind as follows

$$\beta_i = \cos \frac{i\pi}{N-1}, \tag{8}$$

Algorithm 1: Error-locator algorithm.

Input: x_i's, y_i's for $i \in A_{\text{avl}}$, E and K.
Output: Error locations.

Step 1: Find polynomials $P(x) \stackrel{\text{def}}{=} \sum_{i=0}^{K+E-1} P_i x^i$, $Q(x) \stackrel{\text{def}}{=} \sum_{i=0}^{K+E-1} Q_i x^i$ by solving the following system of linear equations

$$P(x_i) = y_i Q(x_i), \qquad \forall i \in A_{\text{avl}}.$$

Step 2: Set $a_i = Q(x_i), \forall i \in A_{\text{avl}}$.
Step 3: Sort a_i's with respect to their absolute values, i.e., $|a_{i_1}| \leq |a_{i_2}| \leq \cdots \leq |a_{i_{N-S}}|$.
Return: $i_1, \cdots, i_E$.

Algorithm 2: ApproxIFER error-locator algorithm.

Input: $f(\tilde{\mathbf{X}}_i)$'s for $i \in A_{\text{avl}}$, β_i's as specified in (8), K, C and E.
Output: The set of indices A_{adv} corresponding to malicious workers.

Set $\mathbf{I} = [\mathbf{0}]_{C\times E}$.
For $l = 1, \cdots, C$

Step 1: Set $P(x) \stackrel{\text{def}}{=} \sum_{j=0}^{K+E-1} P_j x^j$, and $Q(x) \stackrel{\text{def}}{=} \sum_{j=1}^{K+E-1} Q_j x^j + 1$.

Step 2: Solve the system of linear equations provided by

$$P(\beta_i) = f_j(\tilde{\mathbf{X}}_i)Q(\beta_i), \qquad \forall i \in A_{\text{avl}}.$$

to find the coefficients P_j's and Q_j's.
Step 3: Set $a_i = Q(\beta_i), \forall i \in A_{\text{avl}}$.
Step 4: Sort a_i's increasingly with respect to their absolute values, i.e., $|a_{i_1}| \leq \cdots \leq |a_{i_{N-S}}|$.
Step 5: Set $\mathbf{I}[l,:] = [i_1, \cdots, i_E]$.
end
Return: A_{adv}: The set of E most-frequent elements of $\mathbf{I}$.

for $i \in [N-1]$. The i-th worker then computes the prediction on the coded query $\tilde{\mathbf{X}}_i$. That is, the i-th worker computes

$$\tilde{\mathbf{Y}}_i \stackrel{\text{def}}{=} f(\tilde{\mathbf{X}}_i) = f(u(\beta_i)), \tag{9}$$

where $i \in [N-1]$.

ApproxIFER Decoding. When $E = 0$, the decoder waits for the results of the fastest K workers before decoding. Otherwise, in the presence of Byzantine workers, i.e., $E > 0$, the decoder waits for the results of the fastest $2(K+E)$ workers. After receiving the sufficient number of coded predictions, the decoding approach proceeds with the following two steps.

1. **Locating Adversarial Workers.** In presence of Byzantine workers that return erroneous predictions aiming at altering the inference results or even unintentionally, we utilize Algorithm 2 provided below to locate them. The predictions corresponding to these workers can be then excluded in the decoding step. Algorithm 2 runs our proposed error-locator algorithm for rational interpolation in presence of errors provided in Algorithm 1 several times, each time associated to one of the soft labels in the predictions on the coded queries, i.e., $f(\tilde{\mathbf{X}}_i)$'s. At the end, we decide the error locations based on a majority vote on all estimates of the error locations. In Algorithm 2, $f_j(\tilde{\mathbf{X}}_i)$ denotes the j'th coordinate of $f(\tilde{\mathbf{X}}_i)$ which is the soft label corresponding to class j in the prediction on the coded query $f(\tilde{\mathbf{X}}_i)$. Also, C denotes the total number of classes which is equal to the size of $f(\tilde{\mathbf{X}}_i)$'s.
2. **Decoding**. After excluding the erroneous workers, the approximate predictions can be then recovered from the results of the workers who returned correct coded predictions whose indices are denoted by $\mathcal{F}$. Specifically, a rational function r is first constructed as follows

$$r(z) = \frac{1}{\sum_{i\in\mathcal{F}} \frac{(-1)^i}{(z-\beta_i)}} \sum_{i\in\mathcal{F}} \frac{(-1)^i}{(z-\beta_i)} f(\tilde{\mathbf{X}}_i), \tag{10}$$

where $|\mathcal{F}| = K$ when $E = 0$ and $|\mathcal{F}| = 2K + 2E$ otherwise. The approximate predictions denoted by $\hat{\mathbf{Y}}_0, \cdots, \hat{\mathbf{Y}}_{K-1}$ then are recovered as follows

$$\hat{\mathbf{Y}}_j = r(\alpha_j), \tag{11}$$

for all $j \in [K-1]$.

Experiments

In this section, we present extensive experiments to show the effectiveness of ApproxIFER. The experiments are run using PyTorch (Paszke et al. 2019). The latency evaluation experiments are written with MPI4py (Dalcin et al. 2011) and performed on Amazon AWS c5.xlarge instances.

Experiment Setup

We perform extensive experiments on the following datasets and architectures.
Datasets. We run experiments on MNIST (LeCun et al. 1998), Fashion-MNIST (Xiao, Rasul, and Vollgraf 2017), CIFAR (Krizhevsky, Hinton et al. 2009) and ImageNet (Deng et al. 2009) datasets.
Architectures. We consider the following architectures: VGG-16 (Simonyan and Zisserman 2014), DenseNet-161 (Huang et al. 2017), ResNet-18, ResNet-50, ResNet-152 (He et al. 2016) and GoogLeNet (Szegedy et al. 2015).

Some of these architectures such as ResNet-18 and VGG-11 have been considered to evaluate the performance of earlier works for distributed inference tasks. However, the underlying parity model parameters considered in such works are model-specific, i.e., they are required to be trained from the scratch every time one considers a different base model. This imposes a significant burden on the applicability of such approaches in practice due to their compute-intense training requirements, especially for cases where more parity models are needed. In comparison, ApproxIFER is agnostic to the underlying model, and its encoder and decoder do not depend on the employed network architecture as well as the scheme overhead. This enables us to extend our experiments to more complex state-of-the-art models such as ResNet-50, ResNet-152, DenseNet-161, and GoogLeNet.
Baselines. We compare ApproxIFER with the ParM framework (Kosaian, Rashmi, and Venkataraman 2019) in case of tolerating stragglers only. Since we are not aware of any baseline that can handle Byzantine workers in prediction serving systems other than the straightforward replication approach, we compare the performance of ApproxIFER with the replication scheme. Since the accuracy of the replication approach is the same as the base model, we compare the test accuracy of ApproxIFER with that of the base model.
Encoding and Decoding. We employ the encoding algorithm introduced in Section . In the case of stragglers only, the decoding algorithm in Section is used. Otherwise, when some of the workers are Byzantine and return erroneous

results, we first locate such workers by utilizing the error-locator algorithm provided in Algorithm 2, exclude their predictions, and then apply the decoding algorithm in Section to the correct returned results.

Performance Metric. We compare the accuracy of ApproxIFER with the base model accuracy on the test dataset. In the case of stragglers only, we also compare our results with the degraded mode accuracy of ParM. In the second part, we compare the end-to-end latency of ApproxIFER with that of ParM.

Performance Evaluation

Our experiments consist of two parts as follows.

Straggler-Resiliency. In the first part, we consider the case where some of the workers are stragglers and there are no Byzantine workers. We compare our results with the baseline (ParM) and illustrate that our approach outperforms the baseline results for $K = 8, 10, 12$ and $S = 1$. Furthermore, we also illustrate that ApproxIFER can handle multiple stragglers as well by demonstrating its accuracy for $S = 2, 3$. We then showcase the performance of ApproxIFER over several other more complex architectures for $S = 1$ and $K = 12$[1] To generate the results shown in Figure 5 and Figure 6, we used pretrained models on CIFAR-10 dataset[2].

In Figure 5 and Figure 6, we compare the performance of ApproxIFER with the base model, i.e., with one worker node and no straggler/Byzantine workers which is also called the best case, as well as with ParM for $K = 8$ and $K = 12$, respectively, over the test dataset. We considered the ResNet-18 network and observed 19%, 7% and 51% improvement in the degraded mode accuracy compared to ParM for MNIST, Fashion-MNIST and CIFAR-10 datasets, respectively for $K = 8$. For $K = 12$, the accuracy improves by 36%, 17% and 58%, respectively.

We then extend our experiments by considering more stragglers, e.g., $S = 2, 3$, as illustrated in Figure 7. The accuracy loss compared with the best case, i.e., no straggler/Byzantine, is not more than 9.4%, 8% and 4.4% for MNIST, Fashion-MNIST and CIFAR-10 datasets, respectively. Figure 8 demonstrates the performance of ApproxIFER on CIFAR-10 over various state-of-the-art architectures. The accuracy loss for $S = 1$ compared with the best case is 14%, 12%, 14%, 13% and 16% for VGG-16, ResNet-34, ResNet-50, DenseNet-161 and GoogLeNet, respectively.

We also evaluate the accuracy of ApproxIFER over datasets with 100 and 1000 number of classes, namely, CIFAR-100 and ImageNet datasets. We report the top-5 accuracy for these datasets in Figure 9, as in (Kosaian, Rashmi, and Venkataraman 2019), to have a fair comparison.

More recently, a framework known as Coded-InvNet (Dinh and Lee 2021) has been proposed to mitigate stragglers for prediction serving systems based on invertible neural networks (Behrmann et al. 2019) with focus on ResNets. We provide an overview of this work along with the comparison with ApproxIFER in Appendix D, which illustrates that ApproxIFER scales better with K compared to Coded-InvNet.

Byzantine-Robustness. In the second part, we provide our experimental results on the performance of ApproxIFER in the presence of Byzantine workers. We include several results for $K = 12$ and $E = 1, 2, 3$. In our experiments, we select the indices of Byzantine workers at random. These workers add a noise that is drawn from a zero-mean normal Gaussian distribution. Lastly, we illustrate that our algorithm performs well for a wide range of standard deviation σ, namely $\sigma = 1, 10, 100$, thereby demonstrating that our proposed error-locator algorithm performs as promised by the theoretical result regardless of the range of the error values. Moreover, we have also provided experimental results on the accuracy of ApproxIFER under other attack models. The results of these experiments are included in Appendix B.

In Figure 10, we illustrate the accuracy of ApproxIFER with ResNet-18 as the network architecture and for various numbers of Byzantine adversary workers. In this part, we only compare the results with the base model (best case) as there is no other baseline except the straightforward replication scheme. Note that the straightforward replication scheme also attains the best case accuracy, though it requires a significantly higher number of workers, i.e., the number of workers to handle E Byzantine workers in ApproxIFER is $2K + 2E$ whereas it is $(2E+1)K$ in the replication scheme. Our experimental results show that the accuracy loss in ApproxIFER compared with the best case is not more than 6%, 4% and 4.2% for MNIST, Fashion-MNIST and CIFAR-10 dataset, respectively, for up to $E = 3$ malicious workers. These results indicate the success of our proposed algorithm for locating errors in ApproxIFER, as provided in Algorithm 2. Figure 11 demonstrates the accuracy of ApproxIFER in the presence of $E = 2$ Byzantine adversaries to perform distributed inference over several underlying network architectures for the CIFAR-10 dataset. We observe that the accuracy loss is not more than 5% for VGG-16, ResNet-34, ResNet-50, DenseNet-161 and GoogLeNet network architectures when $E = 2$.

Latency Evaluation

Tail Latency. Next, we evaluate the tail latency of ApproxIFER and compare it to ParM as baseline. All experiments are run on Amazon EC2. The underlying network architecture considered is ResNet-18 and the queries are randomly drawn from the MNIST dataset. We use background load to emulate network traffic typical of data analytics workloads. In particular, two workers (EC2 instances) are picked at random to transfer a packet with random size between $10^6 - 2 \times 10^6$ Bytes to each other. Our cluster consists of a master node, one client node and N worker nodes. We ran the inference task on batches of size K 10000 times and reported the mean, median, 99-th, 99.5-th and 99.9-th latency percentiles. We illustrate the results for $K = 7$ in Figure 12.

Our results illustrate that the end-to-end 99-th, 99.5-th and 99.9-th latency of ApproxIFER exceeds that of ParM by no more than 4.3%, 4.4% and 7%, respectively for $K = 7$. We have provided additional comparisons for $K = 9, 10$ in Appendix C. While the latency of ApproxIFER is slightly higher than ParM, ApproxIFER improves the degraded mode

[1]The results of ParM are obtained using the codes available at https://github.com/thesys-lab/parity-models.

[2]The pretrained models are available at https://github.com/huyvnphan/PyTorch_CIFAR10.

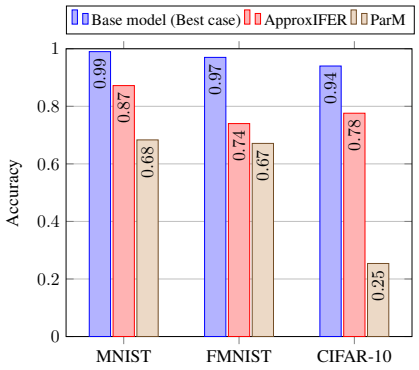

Figure 5: Accuracy of ApproxIFER compared with the best case as well as the degraded mode accuracy of ParM for ResNet-18, $K = 8$ and $S = 1$.

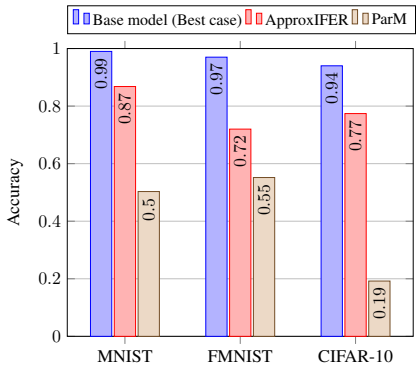

Figure 6: Accuracy of ApproxIFER compared with the best case as well as the degraded mode accuracy of ParM for ResNet-18, $K = 12$ and $S = 1$.

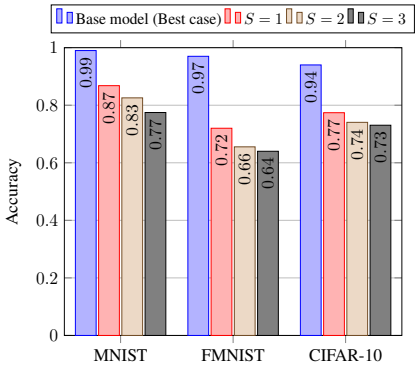

Figure 7: Accuracy of ApproxIFER versus the number of stragglers. The network architecture is ResNet-18, $K = 8$ and $S = 1, 2, 3$.

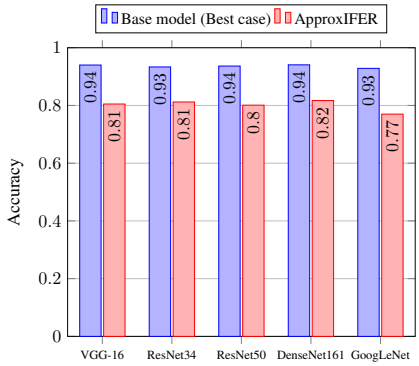

Figure 8: Accuracy of ApproxIFER on CIFAR-10 and with various network architectures for $K = 8$, $S = 1$.

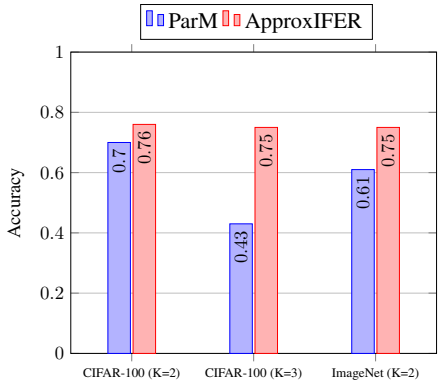

Figure 9: Comparison of the accuracy of ApproxIFER with the degraded mode accuracy of ParM for CIFAR-100 for $K = 2, 3$, $S = 1$ and $E = 0$, and ImageNet datasets for $K = 2$, $S = 1$ and $E = 0$. The network architectures considered are ResNet-50 and ResNet-152 for CIFAR-100 and ImageNet, respectively.

accuracy by up to 58% compared to that of ParM and avoids the significant cost of training S parity models.

Related Works

Replication is widely used for providing straggler resiliency and Byzantine robustness in distributed systems. In the proactive replication approaches, to tolerate S stragglers, the same task is assigned to $S+1$ workers before starting the computation. While such approaches reduce the latency significantly, they incur significant overhead. The reactive replication approaches (Zaharia et al. 2008; Dean and Barroso 2013) avoid such overhead by assigning the same task to other workers only after a deadline is passed as in Hadoop MapReduce (Hadoop 2021). This approach also incurs a significant latency as it waits before reassigning the tasks.

Recently, coding-theoretic approaches have shown great success in mitigating stragglers in distributed computing and machine learning (Lee et al. 2017; Tandon et al. 2017; Yu, Maddah-Ali, and Avestimehr 2017; Ye and Abbe 2018; Wang, Charles, and Papailiopoulos 2019; Soto, Li, and Fan 2019; Narra et al. 2020; Wang, Liu, and Shroff 2019; Soleymani,

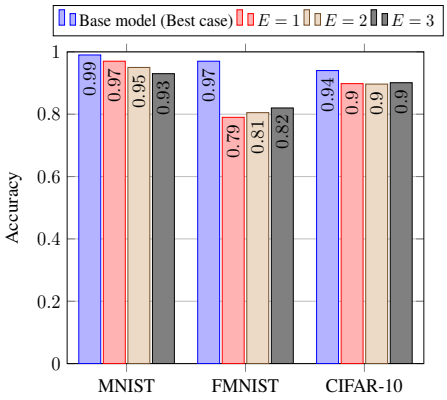

Figure 10: Accuracy of ApproxIFER versus the number of errors on ResNet-18 for $K = 12$, $S = 0$, and $E = 1, 2, 3$.

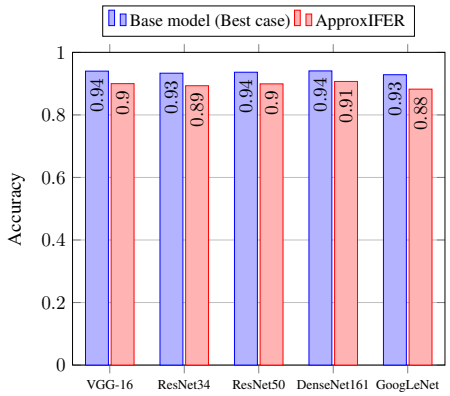

Figure 11: Accuracy of ApproxIFER on CIFAR-10 and various network architectures for $K = 12$, $S = 0$ and $E = 2$.

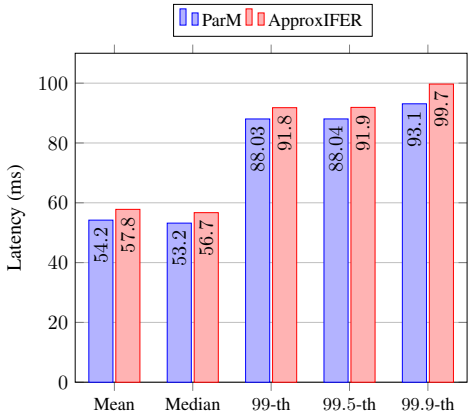

Figure 12: Comparison of the tail latency of ApproxIFER with that of ParM for ResNet-18 for $K = 7$, $S = 1$ and $E = 0$.

Jamali, and Mahdavifar 2021). Such ideas have also been extended to also provide Byzantine robustness and data privacy. Specifically, the coded computing paradigm has recently emerged by adapting erasure coding ideas to design straggler-resilient, Byzantine-robust and private distributed computing systems often involving polynomial-type computations (Yu et al. 2019; Subramaniam, Heidarzadeh, and Narayanan 2019; Soleymani et al. 2021; Tang et al. 2021; So, Guler, and Avestimehr 2020; Sohn et al. 2020; Soleymani, Mahdavifar, and Avestimehr 2021; So et al. 2021). However, many applications involve non-polynomial computations as training and inference of neural networks.

A natural approach to get around the polynomial limitation is to approximate any non-polynomial computations (So, Guler, and Avestimehr 2020). This idea has been leveraged to train a logistic regression model in (So, Guler, and Avestimehr 2020; Ali, So, and Avestimehr 2020; Soleymani, Mahdavifar, and Avestimehr 2020). This approximation approach, however, is not suitable for neural networks as the number of workers needed is proportional to the degree of the function being computed. Motivated by these limitations, learning-based approaches were proposed in (Kosaian, Rashmi, and Venkataraman 2019, 2020) to tackle these challenges. This idea provides the same straggler-resilience as that of the underlying erasure code, and hence decouples the straggler-resilience guarantee from the computation carried out by the system. This is achieved by learning parity models that transform the coded queries to coded predictions.

As discussed, the learning-based approaches do not scale well. This motivated us in this work to explore a different approach based on approximate coded computing. Approximate computing was leveraged before in distributed matrix-matrix multiplication in (Gupta et al. 2018). Moreover, an approximate *coded* computing approach was developed in (Jahani-Nezhad and Maddah-Ali 2021) for distributed matrix-matrix multiplication. More recently, a numerically stable straggler-resilient approximate coded computing approach has been developed in (Jahani-Nezhad and Maddah-ali 2022). In particular, this approach can be used to *approximately* compute arbitrary functions unlike the conventional coded computing techniques. One of the key features of this approach is that it uses rational functions (Berrut and Trefethen 2004) rather than polynomials to introduce coded redundancy which are known to be numerically stable. This approach, however, does not provide robustness against Byzantine workers.

Conclusions

In this paper, we have introduced ApproxIFER, a model-agnostic straggler-resilient, and Byzantine-robust framework for prediction serving systems. Unlike the learning-based approaches, our approach does not require training parity models and can be set to tolerate any number of stragglers and Byzantine workers. Our experiments on MNIST, Fashion-MNIST, CIFAR and ImageNet datasets on various state-of-the-art architectures such as VGG, ResNet, DenseNet, and GoogLeNet show that ApproxIFER improves the degraded mode accuracy by up to 58% compared to the learning-based approaches. An interesting direction is to extend ApproxIFER to preserve the privacy of the data potentially by leveraging trusted execution environments (TEEs) (Tramer and Boneh 2018; Niu, Ali, and Avestimehr 2021). Improving the latency and the accuracy of ApproxIFER through designing systematic codes is also an interesting direction.

Acknowledgements

We thank Jinhyun So, Tuan Dinh and Kangwook Lee for the helpful discussions. This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0053 and FA8750-19-2-1005, ARO award W911NF1810400, NSF grants CCF-1703575, CCF-1763673, CCF–1763348, CCF–1909771 and CCF–1941633, and MLWINS-2002874, ONR Award No. N00014-16-1-2189, and a gift from Intel/Avast/Borsetta via the PrivateAI institute, a gift from Cisco, and a gift from Qualcomm. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

Ali, R. E.; So, J.; and Avestimehr, A. S. 2020. On polynomial approximations for privacy-preserving and verifiable relu networks. *arXiv preprint arXiv:2011.05530*.

Amazon. 2021. Amazon AWS AI. https://aws.amazon.com/machine-learning/. Last accessed: May 2021.

Behrmann, J.; Grathwohl, W.; Chen, R. T.; Duvenaud, D.; and Jacobsen, J.-H. 2019. Invertible residual networks. In *International Conference on Machine Learning*, 573–582. PMLR.

Berrut, J.-P. 1988. Rational functions for guaranteed and experimentally well-conditioned global interpolation. *Computers & Mathematics with Applications*, 15(1): 1–16.

Berrut, J.-P.; and Klein, G. 2014. Recent advances in linear barycentric rational interpolation. *Journal of Computational and Applied Mathematics*, 259: 95–107.

Berrut, J.-P.; and Trefethen, L. N. 2004. Barycentric lagrange interpolation. *SIAM review*, 46(3): 501–517.

Blahut, R. E. 2008. *Algebraic codes on lines, planes, and curves: an engineering approach*. Cambridge University Press.

Bos, L.; De Marchi, S.; and Hormann, K. 2011. On the Lebesgue constant of Berrut's rational interpolant at equidistant nodes. *Journal of Computational and Applied Mathematics*, 236(4): 504–510.

Bos, L.; De Marchi, S.; Hormann, K.; and Sidon, J. 2013. Bounding the Lebesgue constant for Berrut's rational interpolant at general nodes. *Journal of Approximation Theory*, 169: 7–22.

Chaubey, M.; and Saule, E. 2015. Replicated data placement for uncertain scheduling. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, 464–472. IEEE.

Crankshaw, D.; Wang, X.; Zhou, G.; Franklin, M. J.; Gonzalez, J. E.; and Stoica, I. 2017. Clipper: A low-latency online prediction serving system. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 613–627.

Dalcin, L. D.; Paz, R. R.; Kler, P. A.; and Cosimo, A. 2011. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9): 1124–1139.

Dean, J.; and Barroso, L. A. 2013. The tail at scale. *Communications of the ACM*, 56(2): 74–80.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Dinh, T.; and Lee, K. 2021. Coded-InvNet for Resilient Prediction Serving Systems. *arXiv preprint arXiv:2106.06445*.

Gardner, K.; Zbarsky, S.; Doroudi, S.; Harchol-Balter, M.; and Hyytia, E. 2015. Reducing latency via redundant requests: Exact analysis. *ACM SIGMETRICS Performance Evaluation Review*, 43(1): 347–360.

Google. 2021. Google Cloud AI. https://cloud.google.com/products/machine-learning/. Last accessed: May 2021.

Gupta, V.; Wang, S.; Courtade, T.; and Ramchandran, K. 2018. Oversketch: Approximate matrix multiplication for the cloud. In *2018 IEEE International Conference on Big Data (Big Data)*, 298–304. IEEE.

Hadoop. 2021. Apache Hadoop. http://hadoop.apache.org/. Last accessed: May 2021.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Jahani-Nezhad, T.; and Maddah-Ali, M. A. 2021. CodedSketch: A coding scheme for distributed computation of approximated matrix multiplication. *IEEE Transactions on Information Theory*.

Jahani-Nezhad, T.; and Maddah-ali, M. A. 2022. Berrut Approximated Coded Computing: Straggler Resistance Beyond Polynomial Computing. Forthcoming.

Kosaian, J.; Rashmi, K.; and Venkataraman, S. 2019. Parity models: erasure-coded resilience for prediction serving systems. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 30–46.

Kosaian, J.; Rashmi, K.; and Venkataraman, S. 2020. Learning-Based Coded Computation. *IEEE Journal on Selected Areas in Information Theory*, 1(1): 227–236.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. *Technical Report*, University of Toronto.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Lee, K.; Lam, M.; Pedarsani, R.; Papailiopoulos, D.; and Ramchandran, K. 2017. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory*, 64(3): 1514–1529.

Microsoft. 2021. Azure Machine Learning Studio. https://azure.microsoft.com/en-us/services/machine-learning-studio/. Last accessed: May 2021.

Muralee Krishnan, N. K.; Hosseini, S.; and Khisti, A. 2020. Coded Sequential Matrix Multiplication For Straggler Mitigation. *Advances in Neural Information Processing Systems*, 33.

Narra, H. V. K. G.; Lin, Z.; Ananthanarayanan, G.; Avestimehr, S.; and Annavaram, M. 2020. Collage Inference: Using Coded Redundancy for Lowering Latency Variation in Distributed Image Classification Systems. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 453–463. IEEE.

Narra, K. G.; Lin, Z.; Kiamari, M.; Avestimehr, S.; and Annavaram, M. 2019. Slack squeeze coded computing for adaptive straggler mitigation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–16.

Niu, Y.; Ali, R. E.; and Avestimehr, S. 2021. AsymML: An Asymmetric Decomposition Framework for Privacy-Preserving DNN Training and Inference. *arXiv preprint arXiv:2110.01229*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*.

Shah, N. B.; Lee, K.; and Ramchandran, K. 2015. When do redundant requests reduce latency? *IEEE Transactions on Communications*, 64(2): 715–722.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

So, J.; Guler, B.; and Avestimehr, S. 2020. A Scalable Approach for Privacy-Preserving Collaborative Machine Learning. *Advances in Neural Information Processing Systems*, 33.

So, J.; He, C.; Yang, C.-S.; Li, S.; Yu, Q.; Ali, R. E.; Guler, B.; and Avestimehr, S. 2021. LightSecAgg: a Lightweight and Versatile Design for Secure Aggregation in Federated Learning. *arXiv preprint arXiv:2109.14236*.

Sohn, J.-y.; Han, D.-J.; Choi, B.; and Moon, J. 2020. Election coding for distributed learning: Protecting SignSGD against byzantine attacks. *Advances in Neural Information Processing Systems*, 33.

Soleymani, M.; Ali, R. E.; Mahdavifar, H.; and Avestimehr, A. S. 2021. List-decodable coded computing: Breaking the adversarial toleration barrier. *IEEE Journal on Selected Areas in Information Theory*, 2(3): 867–878.

Soleymani, M.; Jamali, M. V.; and Mahdavifar, H. 2021. Coded Computing via Binary Linear Codes: Designs and Performance Limits. *IEEE Journal on Selected Areas in Information Theory*, 2(3): 879–892.

Soleymani, M.; Mahdavifar, H.; and Avestimehr, A. S. 2020. Privacy-preserving distributed learning in the analog domain. *arXiv preprint arXiv:2007.08803*.

Soleymani, M.; Mahdavifar, H.; and Avestimehr, A. S. 2021. Analog lagrange coded computing. *IEEE Journal on Selected Areas in Information Theory*, 2(1): 283–295.

Soto, P.; Li, J.; and Fan, X. 2019. Dual entangled polynomial code: Three-dimensional coding for distributed matrix multiplication. In *International Conference on Machine Learning*, 5937–5945. PMLR.

Subramaniam, A. M.; Heidarzadeh, A.; and Narayanan, K. R. 2019. Collaborative decoding of polynomial codes for distributed computation. In *2019 IEEE Information Theory Workshop (ITW)*, 1–5. IEEE.

Suresh, L.; Canini, M.; Schmid, S.; and Feldmann, A. 2015. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 513–527.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

Tandon, R.; Lei, Q.; Dimakis, A. G.; and Karampatziakis, N. 2017. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, 3368–3376. PMLR.

Tang, T.; Ali, R. E.; Hashemi, H.; Gangwani, T.; Avestimehr, S.; and Annavaram, M. 2021. Verifiable coded computing: Towards fast, secure and private distributed machine learning. *arXiv preprint arXiv:2107.12958*.

Tramer, F.; and Boneh, D. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*.

Wang, H.; Charles, Z.; and Papailiopoulos, D. 2019. Erasurehead: Distributed gradient descent without delays using approximate gradient coding. *arXiv preprint arXiv:1901.09671*.

Wang, S.; Liu, J.; and Shroff, N. 2019. Fundamental limits of approximate gradient coding. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3): 1–22.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Ye, M.; and Abbe, E. 2018. Communication-computation efficient gradient coding. In *International Conference on Machine Learning*, 5610–5619. PMLR.

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

Yu, Q.; Maddah-Ali, M. A.; and Avestimehr, A. S. 2017. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 4406–4416.

Zaharia, M.; Konwinski, A.; Joseph, A. D.; Katz, R. H.; and Stoica, I. 2008. Improving MapReduce performance in heterogeneous environments. In *Osdi*, volume 8, 7.