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

Abstract

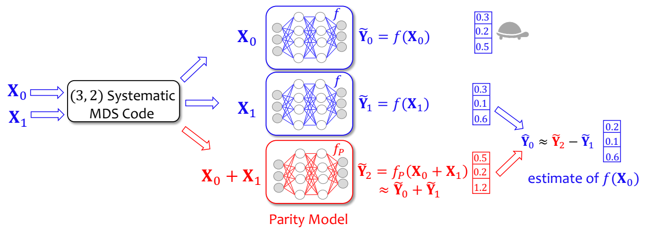
Due to the surge of cloud-assisted AI services, the problem of designing resilient prediction serving systems that can effectively cope with stragglers/failures and minimize response delays has attracted much interest. The common approach for tackling this problem is replication which assigns the same prediction task to multiple workers. This approach, however, is very 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 in order 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 (mostly one) number of stragglers. Moreover, ParM does not handle Byzantine adversarial workers. We propose a different approach, named Approximate Coded Inference (ApproxIFER), that does not require training of 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 also show significant accuracy improvement by up to 58% over the parity model approaches.

1 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 [1, 3, 4, 34]. More specifically, prediction serving systems host complex machine learning models and respond to the inference queries of the clients 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 worker nodes in the system each having an instance of the deployed model [15]. Such systems often 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 [43, 2, 5, 16, 36, 19, 14]. Replication-based systems, however, entail 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 [29, 17, 52, 53, 33, 31, 41]. The traditional coding-theoretic approaches, known as the coded computing approaches, are usually 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 has been proposed in [26]. In this approach, the prediction queries are first 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 can be 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.



$\mathbf{X}_0$
$\mathbf{X}_1$
$(3,2)$ Systematic MDS Code
$\mathbf{X}_0$
f
$\tilde{\mathbf{Y}}_0 = f(\mathbf{X}_0)$
0.3
0.2
0.5
$\mathbf{X}_1$
f
$\tilde{\mathbf{Y}}_1 = f(\mathbf{X}_1)$
0.3
0.1
0.6
$\mathbf{X}_0 + \mathbf{X}_1$
f_P
$\tilde{\mathbf{Y}}_2 = f_P(\mathbf{X}_0 + \mathbf{X}_1) \approx \tilde{\mathbf{Y}}_0 + \tilde{\mathbf{Y}}_1$
0.5
0.2
1.2
Parity Model
$\hat{\mathbf{Y}}_0 \approx \tilde{\mathbf{Y}}_2 - \tilde{\mathbf{Y}}_1$
0.2
0.1
0.6
estimate of $f(\mathbf{X}_0)$

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 $f_P(\mathbf{X}_0 + \mathbf{X}_1)$.

These parity models, however, depend on the model hosted by the cloud and are suitable for tolerating one straggler and

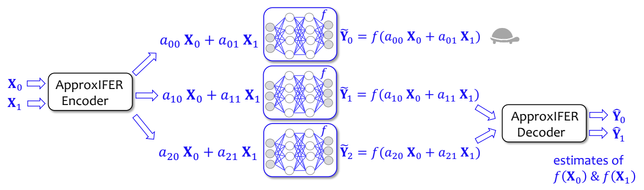

$\mathbf{X}_0$
$\mathbf{X}_1$
ApproxIFER Encoder
$a_{00}\,\mathbf{X}_0 + a_{01}\,\mathbf{X}_1$
$a_{10}\,\mathbf{X}_0 + a_{11}\,\mathbf{X}_1$
$a_{20}\,\mathbf{X}_0 + a_{21}\,\mathbf{X}_1$
f
$\tilde{\mathbf{Y}}_0 = f(a_{00}\,\mathbf{X}_0 + a_{01}\,\mathbf{X}_1)$
$\tilde{\mathbf{Y}}_1 = f(a_{10}\,\mathbf{X}_0 + a_{11}\,\mathbf{X}_1)$
$\tilde{\mathbf{Y}}_2 = f(a_{20}\,\mathbf{X}_0 + a_{21}\,\mathbf{X}_1)$
ApproxIFER Decoder
$\hat{\mathbf{Y}}_0$
$\hat{\mathbf{Y}}_1$
estimates of $f(\mathbf{X}_0)$ & $f(\mathbf{X}_1)$

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.

handling a small number of queries (typically less than 4). Moreover, they require retraining whenever they are used with a new cloud model. In this work, we take a different approach leveraging approximate coded computing techniques [23] to design scalable, straggler-resilient and Byzantine-robust prediction serving systems. Our approach relies on rational interpolation techniques [6] 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 encoding and the decoding procedures of ApproxIFER do not depend on the model depolyed by the cloud. 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 tolerate 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 MNIST, Fashion-MNIST, and CIFAR-10 datasets on VGG, ResNet, DenseNet, and GoogLeNet architectures which show that ApproxIFER improves the prediction 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.

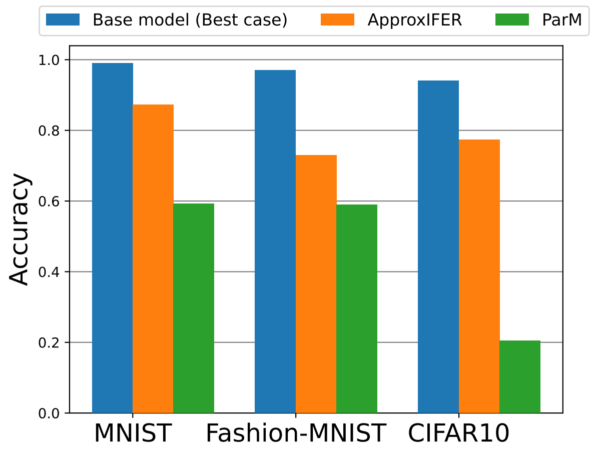


Figure 3: Comparison of the accuracy of ApproxIFER with the base model test accuracy for ResNet-18 and ParM for for $K = 10$, $S = 1$ and $E = 0$.

Organization. The rest of this paper is organized as follows. We describe the problem setting in Section 2. Then, we describe ApproxIFER in Section 3. In Section 4, we present our extensive experimental results. In Section 5, we discuss the closely-related works. Finally, we discuss some concluding remarks and the future research directions in Section 6.

2 Problem Setting

System Architecture. We consider a prediction serving system with $N + 1$ 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 [26], 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 being resilient to any S stragglers and robust to any E Byzantine workers.

3 ApproxIFER Algorithm

In this section, we present our proposed protocol based on leveraging approximate coded computing. 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 [7]. On the other hand, rational interpolation is known to be extremely stable and can lead to faster convergence compared to polynomial interpolation [6]. This motivated a recent work to leverage rational functions rather than polynomials to design straggler-resilient

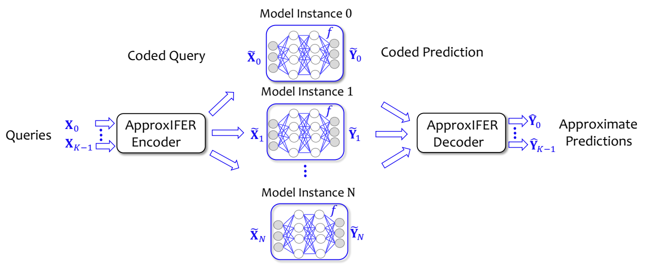


Figure 4: An illustration of the architecture of ApproxIFER. In ApproxIFER, all workers have the model deployed by the system f, no parity models are required and only an encoder and a decoder are added on top of the conventional replication-based prediction serving systems. The K input queries $\mathbf{X}_0, \cdots, \mathbf{X}_{K-1}$ 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.

distributed training algorithms [23]. ApproxIFER also leverages rational functions and rational interpolation. We provide a very brief background about rational functions next.

Rational Interpolation Background. Consider a function f, $n+1$ distinct points $a \le x_0 < x_1, \cdots < x_n \le b$ and the corresponding evaluations of f at these points denoted by $f_0, f_1, \cdots, f_{n-1}, f_n$. Berrut's rational interpolant of f is then defined as follows [6]:

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

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

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

for $i \in [n]$. Berrut's rational interpolant has several useful properties as it has no pole on the real line [6] and it is extremely well-conditioned [11, 12]. It also converges with rate $O(h)$, where $h \overset{\text{def}}{=} \max_{0 \le i \le n-1} x_{i+1} - x_i$ [18].

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 $N - S + 1$ available evaluations of $r(x)$ over x_i's, for some $S \ge 1$ that denote the number of stragglers in the context of our system model. Let A_{adv}, with $|A_{\text{adv}}| \le E$, denote 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 $N - S - E + 1$ 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 [10]. 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 below.

```
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 $P(x) \overset{\text{def}}{=} \sum_{i=0}^{K+E-1} P_i x^i$, $Q(x) \overset{\text{def}}{=} \sum_{i=0}^{K+E-1} Q_i x^i$ by solving the following system of linear equations:
    $P(x_i) = y_i Q(x_i), \quad \forall i \in A_{\text{avl}}.$
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., $|a_{i_1}| \le |a_{i_2}| \le \cdots |a_{i_{N-S+1}}|$.
Return: $i_1, \cdots, i_E$.
```

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 $N - S + 1$. In order to guarantee the existence of a non-trivial solution, we must have

$$N \ge 2K + 2E + S - 1. \tag{3}$$

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 $N+1$ coded queries, denoted by $\tilde{\mathbf{X}}_i$, for $i \in [N]$, 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 $N = K + S - 1$ which corresponds to an overhead of $\frac{K+S}{K}$. Otherwise, $N = 2(K+E) + S - 1$ 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. 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

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

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

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

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

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

where $i \in [N]$.

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.

```
Algorithm 2: ApproxIFER error-locator algorithm.
Input: $f(\tilde{\mathbf{X}}_i)$'s for $i \in A_{\text{avl}}$, $\beta_i$'s as specified in (8), $K$, C and $E$.
Output: The set of indices $A_{\text{adv}}$ corresponding to malicious workers.

Set $\mathbf{I} = [\mathbf{0}]_{C \times E}$.
For $j = 1, \cdots, C$
    Step 1: Set $P(x) \overset{\text{def}}{=} \sum_{j=0}^{K+E-1} P_j x^j$, and $Q(x) \overset{\text{def}}{=} \sum_{j=1}^{K+E-1} Q_j x^j + 1$.
    Step 2: Solve the system of linear equations provided by
        $P(\beta_i) = f_j(\tilde{\mathbf{X}}_i) Q(\beta_i), \quad \forall i \in A_{\text{avl}}.$
        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., $|a_{i_1}| \leq \cdots \leq |a_{i_{N-S+1}}|$.
    Step 5: Set $\mathbf{I}[j,:] = [i_1, \cdots, i_E]$.
end
Return: $A_{\text{adv}}$: The set of $E$ most-frequent elements of $\mathbf{I}$.
```

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

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

where $|\mathcal{F}| = K$ when $E = 0$ and $|\mathcal{F}| = 2K + E$ 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 $j \in [K-1]$.

4 Experiments

In this section, we present several experiments to show the effectiveness of ApproxIFER.

4.1 Experiment Setup

More specifically, we perform extensive experiments on the following dastsests and architectures. All experiments are run using PyTorch [35] using a MacBook pro with 3.5 GHz dual-core Intel core i7 CPU. The code for implementing ApproxIFER is provided as a supplementary material.

Datasets. We run experiments on MNIST [28], Fashion-MNIST [49] and CIFAR-10 [27] datasets.

Architectures. We consider the following network architectures: VGG-16 [37], DenseNet-161 [22], ResNet-18, ResNet-50, ResNet-152 [21], and GoogLeNet [44].

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 computational heavy training requirements, especially for cases where more than one models parities 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 [26] in case of tolerating stragglers only. Since we are not aware of any baseline that can handle Byzantine workers in the literature 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 replication of 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 3. In the case of stragglers only, the decoding algorithm in Section 3 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 3 to the correct returned results.

Performance metric. We compare the accuracy of predictions in ApproxIFER with the base model accuracy on the test dataset. In the case of stragglers only, we also compare our results with the accuracy of ParM.

4.2 Performance evaluation

Our experiments consist of two parts as follows.
Straggler-Resilience. 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 ResNet-18 network architecture and for $K = 8$ observed 19%, 7% and 51% improvement in the accuracy compared with ParM for the image classification task over MNIST, Fashion-MNIST and CIFAR10 datasets, respectively. For $K = 12$, the accuracy improve by 36%, 17% and 58%, respectively.

We then extend our experiments by considering more stragglers, e.g., $S = 2, 3$, and the results are 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 CIFAR10 datasets, respectively. Figure 8 demonstrates the performance of ApproxIFER for image classification task over CIFAR-10 dataset over various state-of-the-art network 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.

Byzantine-Robustness. In the second part, we provide our experimental results on the performance of ApproxIFER in the presence of Byzantine adversary workers. We include several results for $K = 12$ and $E = 1, 2, 3$. In our experiments, the indices of Byzantine workers are determined 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. The results for this experiment are included in Appendix B.

In Figure 9, 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 10 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 $E = 2$.

5 Related Works

Replication is the most widely used technique for providing straggler resiliency and Byzantine robustness in distributed systems. In this technique, the same task is assigned to multiple workers either proactively or reactively. In the proactive approaches, to tolerate S stragglers, the same task is assigned

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

[2]The models are available at https://github.com/huyvnphan/PyTorch_ CIFAR10.

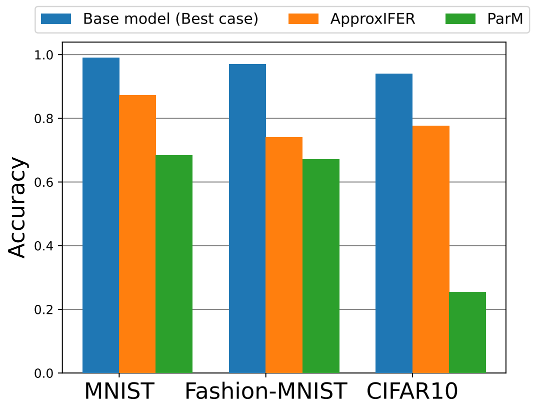


Figure 5: Accuracy of ApproxIFER compared with the best case as well as ParM for ResNet-18, $K = 8$ and $S = 1$.

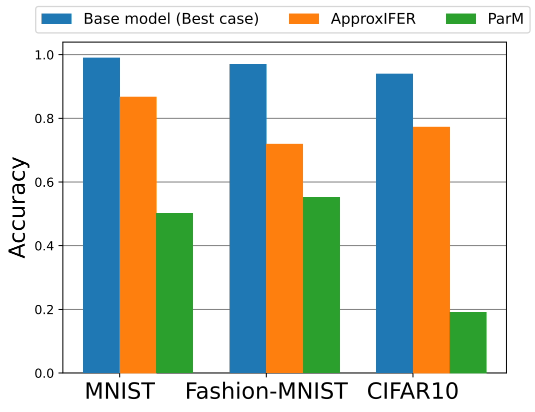


Figure 6: Accuracy of ApproxIFER compared with the best case as well as ParM for ResNet-18, $K = 12$ and $S = 1$.

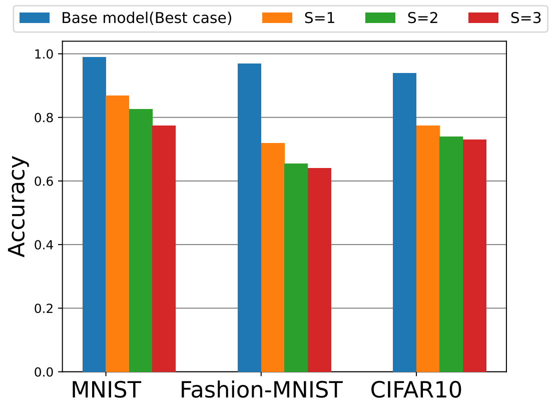


Figure 7: Accuracy of ApproxIFER versus the number of stragglers. The network architecture is ResNet-18, $K = 8$ and $S = 1, 2, 3$.

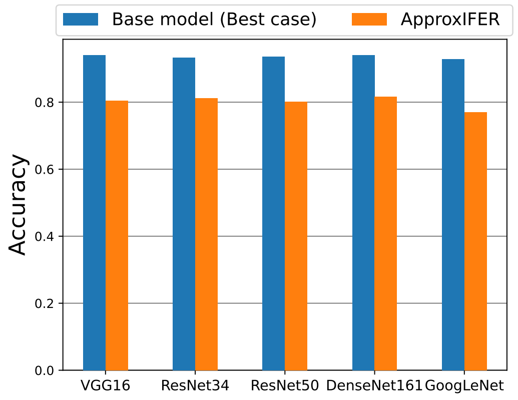


Figure 8: Accuracy of ApproxIFER for image classification over CIFAR-10 and with various network architectures for $K = 8$, $S = 1$.

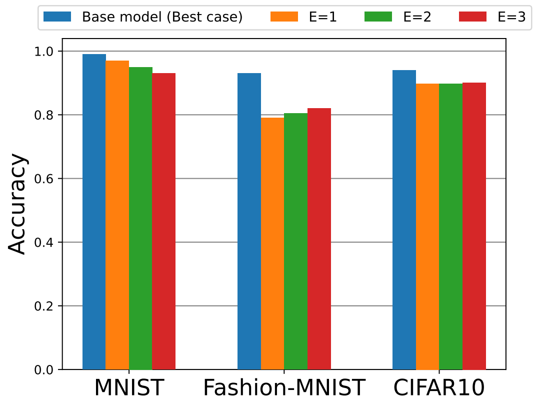


Figure 9: Accuracy of ApproxIFER versus the number of errors on ResNet-18 for $K = 12$, $S = 0$, and $E = 1, 2, 3$.

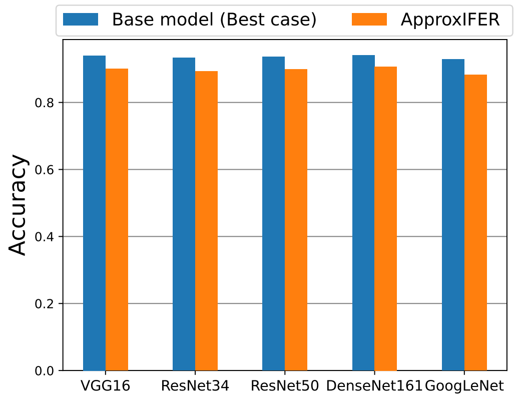


Figure 10: Accuracy of ApproxIFER for image classification over CIFAR-10 and various network architectures for $K = 12$, $S = 0$ and $E = 2$.

to $S + 1$ workers before starting the computation. While such approaches reduce the latency significantly, they incur significant overhead. The reactive approaches [54, 16] avoid such overhead by assigning the same task to other workers only after a deadline is passed as in Hadoop MapReduce [2]. This approach also incurs a significant latency cost as it has to wait before reassigning the tasks.

Recently, coding-theoretic approaches have shown great success in mitigating stragglers in distributed computing and machine learning [29, 45, 53, 51, 47, 41, 32, 17, 48]. Such ideas have also been extended to not only provide straggler resiliency, but also Byzantine robustness and data privacy. Specifically, the coded computing paradigm has recently emerged by adapting erasure coding based ideas to design straggler-resilient, Byzantine-robust and private distributed computing systems often involving polynomial-type computations [50, 52, 42, 40, 46, 38, 39, 30]. However, many applications involve non-polynomial computations such as distributed training and inference of neural networks.

A natural approach to get around the polynomial limitation is to approximate any non-polynomial computations. This idea has been leveraged to train a logistic regression model in [38]. 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 and also the number of queries. Motivated by mitigating these limitations, a learning-based approach was proposed in [26] to tackle these challenges in prediction serving systems. 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 a parity model known as ParM that transforms the coded queries to coded predictions.

As we discussed, the learning-based approaches do not scale well. This motivates us in this work to explore a different approach based on approximate coded computing [24, 23]. Approximate computing was leveraged before in distributed matrix-matrix multiplication in [20]. Moreover, an approximate *coded* computing approach was developed in [24] for distributed matrix-matrix multiplication. More recently, a numerically stable straggler-resilient approximate coded computing approach has been developed in [23]. In particular, this approach is not restricted to polynomials and 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 [8] rather than polynomials to introduce coded redundancy which are known to be numerically stable. This approach, however, does not provide robustness against Byzantine workers. Finally, this approach has been leveraged in distributed training of LeNet-5 using MNIST dataset [28] and resulted in an accuracy that is comparable to the replication-based strategy.

6 Conclusions

In this work, we have introduced ApproxIFER, a model-agnostic straggler-resilient, and Byzantine-robust framework for prediction serving systems. The key idea of ApproxIFER is that it encodes the queries carefully such that the desired predictions can be recovered efficiently in the presence of both stragglers and Byzantine workers. Unlike the learning-based approaches, our approach does not require training any parity models, can be set to tolerate any number of stragglers and Byzantine workers efficiently. Our experiments on the MNIST, the Fashion-MNIST and the CIFAR-10 datasets on various architectures such as VGG, ResNet, DenseNet, and GoogLeNet show that ApproxIFER improves the prediction accuracy by up to 58% compared to the learning-based approaches. An interesting future direction is to extend ApproxIFER to be preserve the privacy of data.

References

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

[2] ???? Apache Hadoop. http://hadoop.apache.org/. Last accessed: May 2021.

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

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

[5] Ananthanarayanan, G.; Ghodsi, A.; Shenker, S.; and Stoica, I. 2013. Effective straggler mitigation: Attack of the clones. In *10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)*, 185–198.

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

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

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

[9] Blackburn, S. R. 1997. Fast rational interpolation, Reed-Solomon decoding, and the linear complexity profiles of sequences. *IEEE Transactions on Information Theory*, 43(2): 537–548.

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

[11] 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.

[12] 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.

[13] Boyer, B.; and Kaltofen, E. L. 2014. Numerical linear system solving with parametric entries by error correction. In *Proceedings of the 2014 Symposium on Symbolic-Numeric Computation*, 33–38.

[14] 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.

[15] 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.

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

[17] Dutta, S.; Cadambe, V.; and Grover, P. 2016. " Short-Dot" computing large linear transforms distributedly using coded short dot products. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2100–2108.

[18] Floater, M. S.; and Hormann, K. 2007. Barycentric rational interpolation with no poles and high rates of approximation. *Numerische Mathematik*, 107(2): 315–331.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] Jahani-Nezhad, T.; and Maddah-Ali, M. A. 2020. Berrut Approximated Coded Computing: Straggler Resistance Beyond Polynomial Computing. *arXiv preprint arXiv:2009.08327*.

[24] 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*.

[25] Kaltofen, E.; and Pernet, C. 2013. Cauchy interpolation with errors in the values. *Submitted manuscript*, 13.

[26] 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.

[27] Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

[28] 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.

[29] 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.

[30] Mallick, A.; Chaudhari, M.; and Joshi, G. 2019. Fast and efficient distributed matrix-vector multiplication using rateless fountain codes. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 8192–8196. IEEE.

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

[32] 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.

[33] 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.

[34] Olston, C.; Fiedel, N.; Gorovoy, K.; Harmsen, J.; Lao, L.; Li, F.; Rajashekhar, V.; Ramesh, S.; and Soyke, J. 2017. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*.

[35] 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*.

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

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

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

[39] 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.

[40] Soleymani, M.; Ali, R. E.; Mahdavifar, H.; and Avestimehr, A. S. 2021. List-Decodable Coded Computing: Breaking the Adversarial Toleration Barrier. *arXiv preprint arXiv:2101.11653*.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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*.

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

[48] 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.

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

[50] Yang, Y.; Grover, P.; and Kar, S. 2017. Coded distributed computing for inverse problems. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 709–719.

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

[52] 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.

[53] 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.

[54] 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.

A Rational Interpolation in the Presence of Errors

In this section, we provide an algebraic method to interpolate a rational function using its evaluations where some of them are erroneous. Let A_{adv} denote the set of indices corresponding to erroneous evaluations and $|A_{\text{adv}}| \leq E$. The proposed algorithm is mainly inspired by the well-known Berlekamp–Welch (BW) decoding algorithm for Reed-Solomon codes in coding theory [10]. This algorithm enables polynomial interpolation in the presence of erroneous evaluations. Our proposed algorithm extends the BW algorithm to rational functions. Extending the BW algorithm to interpolate rational functions in presence of erroneous evaluations has been studied in the literature, e.g., see [13, 25, 9]. However, we propose a simple yet powerful algorithm that guarantees successful recovery under a slightly different conditions. It is worth noting that no assumption is made on the distribution of the error in this setup and the algorithm finds the rational function and the error locations successfully as long as the number of errors is less than a certain threshold.

Let N denote the total number of evaluation points. Let also S and E denote the number of points over which the evaluations of f are unavailable (erased) and erroneous (corrupted), respectively. Consider the following polynomial:

$$\Lambda(x) = \prod_{i \in A_{\text{adv}}} (1 - \frac{x}{x_i}). \tag{12}$$

This polynomial is referred to as the *error-locator* polynomial as its roots are the evaluation points corresponding to the erroneous evaluations. Suppose that the following rational function $r(x)$ is given:

$$r(x) = \frac{p_0 + p_1 x + \cdots + p_{K-1} x^{K-1}}{q_0 + q_1 x + \cdots + q_{K-1} x^{K-1}}. \tag{13}$$

Let $x_0, \cdots, x_N$ denote the evaluation points. Let A_{avl} denote the set of indices corresponding to $N - S + 1$ available evaluations of $r(x)$ over x_i's, for some $S \geq 1$ that denote the number of stragglers in the context of our system model. 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 $N - S - E + 1$ indices $i \in A_{\text{avl}}$. Then we have

$$r(x_i)\Lambda(x_i) = y_i \Lambda(x_i), \qquad \forall i \in A_{\text{avl}}, \tag{14}$$

which obviously holds for any i with $y_i = r(x_i)$. Otherwise, when the evaluation is erroneous, i.e., $i \in A_{\text{adv}}$, then we have $\Lambda(x_i) = 0$ implying that (14) still holds. Let

$$P(x) \overset{\text{def}}{=} p(x)\Lambda(x) = \sum_{i=0}^{K+E-1} P_i x^i, \tag{15}$$

and

$$Q(x) \overset{\text{def}}{=} q(x)\Lambda(x) = \sum_{i=0}^{K+E-1} Q_i x^i. \tag{16}$$

Plugging in (15) and (16) into (14) results in

$$P(x_i) = y_i Q(x_i), \qquad \forall i \in A_{\text{avl}}. \tag{17}$$

The equations in (17) form a homogeneous system of linear equations whose unknown variables are $P_0, \cdots, P_{K+E-1}, Q_0, \cdots, Q_{K+E-1}$. Then the number of unknown variables is $2(K+E)$ and the number of equations is $N - S + 1$. In order to guarantee finding a solution to the set of equations provided in (17) the number of variables we must have

$$N \geq 2K + 2E + S - 1. \tag{18}$$

This guarantees the existence of a solution to $P(x)$ and $Q(x)$. since the underlying system of linear equations is homogeneous. After determining the polynomials $P(x)$ and $Q(x)$, defined in (15) and (16), respectively, the rational function $r(x)$ is determined by dividing $P(x)$ by $Q(x)$, i.e., $r(x) = \frac{p(x)}{q(x)} = \frac{P(x)}{Q(x)}$. We summarize the proposed algorithm in Algorithm 3.

Algorithm 3: BW-type rational interpolation in presence of errors.

Input: x_i's, y_i's for $i \in A_{\text{avl}}$ and K.
Output: The rational function $r(x)$.
Step 1: Find the polynomials $P(x)$ and $Q(x)$, as described in (15) and (16), respectively, such that

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

Step 2: Set $r(x) = \frac{P(x)}{Q(x)}$.
Return: $r(x)$.

The condition (18) guarantees that the step 1 of Algorithm 3 always finds polynomials $P(x)$ and $Q(x)$ successfully. Next, it is shown that $r(x) = \frac{P(x)}{Q(x)}$ in the following theorem.

Theorem 1. *The rational function returned by Algorithm 3 is equal to $r(x)$, as long as (18) holds and $Q(x) \neq 0$.*

Proof. Recall that $r(x) = \frac{p(x)}{q(x)}$. Let $N(x) \overset{\text{def}}{=} P(x)q(x) - Q(x)p(x)$. Then, $\deg(N(x)) \leq 2K + E - 2$. This implies that $N(s)$ has at most $2K + E - 1$ roots.

On the other hand, note that $y_i = r(x_i)$ for at least $N - S - E + 1$ points since at most E out of $N - S + 1$ available evaluations are erroneous. Then, $P(x_i) = r(x_i)Q(x_i) = \frac{p(x_i)}{q(x_i)}Q(x_i)$ which implies $P(x_i)q(x_i) - Q(x_i)p(x_i) = N(x_i) = 0$ in at least $N - S - E + 1$ points. Hence, $N(x) = 0$ if $2K + E - 1 \leq N - S - E + 1$ which implies $r(x) = \frac{p(x)}{q(x)} = \frac{P(x)}{Q(x)}$. □

Algorithm 3 is prone to numerical issues in practice due to round-off errors. For implementation purposes, instead of dividing $P(x)$ by $Q(x)$ in step 2 of Algorithm 3, we evaluate $Q(x)$ over x_i's for all $i \in A_{\text{avl}}$ and declare the indices corresponding to x_i's having the least E absolute values as error locations. The rational function $r(x)$ then can be interpolated by excluding the erroneous evaluations. The error-locator algorithm we use in our implementations is provided in Algorithm 1.

B Further Experiments

In this section, we illustrate that our algorithm performs well for a wide range of standard deviation σ. Our experimental results demonstrate that our proposed error-locator algorithm performs as promised by the theoretical result regardless of the range of the error values. In Figure 11, we illustrate the accuracy of ApproxIFER with ResNet-18 as the network architecture for the image classification task over MNIST and Fashion-MNIST datasets. The results for $\sigma = 1, 10, 100$ are compared with eachother. It illustrates the proposed error-locator algorithm performs well for a wide range of σ.

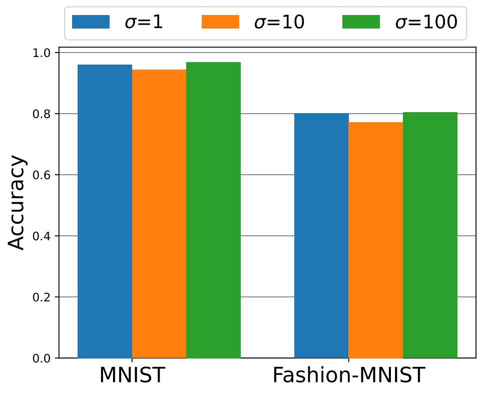


Figure 11: Comparison of the accuracy of ApproxIFER for several values of noise standard deviation σ. Other parameters are $K = 8$, $S = 0$ and $E = 2$. The underlying network architecture is ResNet-18 and the experiment is run for MNIST and Fashion-MNIST datasets.

C Comparison with ParM: Average Case versus Worst Case

We have compared the accuracy of ApproxIFER with ParM for the worst-case scenario in all reported results in Section 4. For ApproxIFER the worst-case and average-case scenarios are basically the same as all queries are regarded as parity queries. For ParM, the worst-case means that one of uncoded predictions is always unavailable. Note that in $\frac{1}{K+1}$ fraction of the times, ParM has access to all uncoded predictions and hence its accuracy is equal to the base model accuracy. In particular, the average-case accuracy of ParM is greater than the worst-case accuracy by at most $\frac{100}{9}\% \sim 11\%$ since $K \geq 8$ in all of our experiments. Hence, ApproxIFER still outperforms ParM up to 47% in terms of average-case accuracy of the predictions.