THE MORE THE MERRIER: NUMERICALLY ACCURATE CODED COMPUTING FOR MACHINE LEARNING INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Coded distributed computing mitigates stragglers in machine learning inference, but modern neural network architectures consist of both polynomial and non-polynomial computations. For polynomial components, exact recovery is possible at a recovery threshold D, but the resulting decoding matrix can be ill-conditioned, leading to numerical instability under finite-precision arithmetic. Existing frameworks address this issue either by directly optimizing the decoding matrix or by carefully selecting the evaluation points to reduce its condition number. Despite these efforts, the condition number of the decoding matrix can remain large at the recovery threshold D. For general non-polynomial functions, approximation-based coded computing avoids the difficulty of exact reconstruction but suffers from approximation error. Classical exact-recovery schemes typically discard worker responses beyond the recovery threshold, leaving available computational redundancy underutilized. In this work, we improve recovery accuracy in both regimes by exploiting $N > D$ available worker responses. For polynomial subnetworks, we introduce **ChebyLS**, an overdetermined Chebyshev least-squares decoder that utilizes all $N > D$ responses to lower the condition number. We derive a probabilistic condition-number bound that sharpens a general coherence-based row-sampling result by exploiting deterministic structure of the full Chebyshev-Vandermonde matrix and concentration of the missing workers. For non-polynomial subnetworks, we propose **RepApprox**, a hybrid Repetition-Approximation strategy that combines replicated exact evaluations with approximation-based coded computing. With K tasks replicated c times across M servers, we show that the probability of exact recovery with N responses is at least $1 - K(1 - N/M)^c$, where a larger N increases the probability of achieving exact inference, with bounded-error approximation as a fallback when a repetition group is lost. We integrate ChebyLS and RepApprox into the heterogeneous Deep & Cross Network v2 architecture, applying them to its polynomial Cross and non-polynomial Deep branches, respectively. Experiments under finite-precision worker computation demonstrate improved end-to-end inference accuracy by exploiting responses beyond the recovery threshold.

1 INTRODUCTION

Machine learning inference services often evaluate the same model on many inputs under stringent latency requirements. To handle this massive scale and meet stringent latency requirements, these computations are typically distributed across multiple worker nodes. However, such distributed architectures are highly vulnerable to *stragglers* (slow or unresponsive nodes) that bottleneck the entire system (Dean & Barroso, 2013). For instance, empirical measurements on AWS Lambda showed that across 3,000 distributed matrix multiplications, tail latencies severely degraded performance; although the median response time was 40 seconds, the 95th percentile of workers took 100 seconds, and the slowest nodes dragged the maximum latency out to 375 seconds (Gupta et al., 2018). When these computations are distributed across multiple

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

workers, coded distributed computing provides a mechanism for mitigating the effect of stragglers. Specifically, instead of assigning uncoded inputs directly to individual workers, the master encodes multiple inputs and asks each worker to evaluate the target computation on a coded input. The desired computation results can subsequently be recovered from a sufficiently large subset of the worker responses. Representative works include Yu et al. (2017; 2019); Zhu & Li (2022); Kim et al. (2024), with broader surveys available in Li & Avestimehr (2020).

Typical neural network architectures exhibit diverse algebraic structures, ranging from strictly polynomial to entirely non-polynomial, or a mixture of both. For instance, Π-Nets (Deep Polynomial Neural Networks) (Chrysos et al., 2020) rely on polynomial feature interactions, serving as a purely polynomial inference model. By contrast, standard Convolutional Neural Networks (CNNs) (Goodfellow et al., 2016) operate as non-polynomial architectures, as their degree-1 linear convolutions are tightly coupled with non-linear activation functions and max-pooling operations. Demonstrating a mixed architecture, Deep & Cross Networks (DCNs) (Wang et al., 2017; 2021) combine both elements, consisting of two subnetworks, one subnetwork performs exact polynomial computations, while the other applies non-polynomial operations. Earlier works on coded distributed computing focus on matrix multiplication and polynomial computation over finite-field domain (Yu et al., 2017; Dutta et al., 2020; Yu et al., 2019; Zhu & Li, 2022; Kim et al., 2024) and real-number domain (Subramaniam et al., 2020; Soleymani et al., 2021; Fahim & Cadambe, 2021). To broaden its applicability, recent advancements such as Berrut Approximated Coded Computing (BACC) (Jahani-Nezhad & Maddah-Ali, 2023) and Basis Spline Assisted Coded Computing (BSACC) (Borah et al., 2026) abandon the strict polynomial assumption and move towards approximation-based coded computing which can be applied directly to the entire model.

Recognizing the diverse algebraic structures of modern neural networks, we tailor our coded computing strategies to match the target architecture. For pure polynomial architectures or polynomial subnetworks, we aim to improve the recovery accuracy of exact polynomial coded computing by reducing the condition number of the decoding matrix. For pure non-polynomial architectures, we aim to improve the accuracy of approximation-based frameworks by designing a new encoding-decoding scheme which allows the master node to obtain exact results algebraically with high probability. Furthermore, for mixed architectures, we evaluate two options: (1) applying our improved approximation-based framework uniformly across the entire mixed architecture, or (2) applying our improved polynomial and approximation-based schemes separately to the polynomial and non-polynomial subnetworks, respectively.

The primary challenge in the exact polynomial regime stems from the transition from traditional polynomial coded computing which operates on finite field elements to real-number domain. In traditional polynomial coded computing, exact recovery can be achieved as long as recovery threshold is met. That is, if the target computation is a polynomial of degree $D-1$, exactly D worker responses are sufficient for exact polynomial reconstruction. However, modern machine learning inference operates over the real-number domain using finite-precision floating-point arithmetic. In this regime, with exactly D responses, we solve a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$ with a square matrix $\boldsymbol{A}$. At this exact recovery threshold, the condition numbers of the resulting decoding matrices remain problematically high (see Figure 1). Consequently, the floating-point error generated by the worker servers is drastically amplified during the master node's decoding phase, severely degrading the accuracy of the reconstructed polynomial.

A notable advancement in addressing numerical stability for real-valued polynomial coded computing was proposed by Fahim & Cadambe (2021), who replaced the monomial basis with orthogonal Chebyshev polynomials and analyzed the condition numbers of the resulting Chebyshev–Vandermonde decoding matrices. Although this improves numerical stability, decoding at the recovery threshold still requires solving a square linear system whose underlying matrix condition number remains large (see Figure 1). Other works have pursued numerical stability through alternative coding architectures and decoding strategies (Tang et al., 2018; Subramaniam et al., 2020; Soleymani et al., 2021; Das et al., 2021). By contrast, we retain the Cheby-

shev polynomial coding framework and ask whether *additional worker responses arriving after the classical recovery threshold* can themselves be exploited to improve numerical stability.

In the non-polynomial regime, the primary challenge lies in the difficulty of reconstructing general mathematical functions. While it is possible to apply polynomial approximations to commonly used non-linear activation functions, such as ReLU and $\mathrm{Softmax}$, these approximations are accurate only when the inputs to these specific functions are bounded. Imposing such tight constraints on a model's internal pre-activation values is impractical, rendering traditional polynomial decoders inapplicable. To navigate this limitation, frameworks such as BACC and BSACC adopt an approximation-based approach. Rather than attempting exact function reconstruction, these methods approximate the general target function whose approximation error is bounded. For example, BACC (Jahani-Nezhad & Maddah-Ali, 2023) uses Barycentric rational interpolation, LeTCC (Moradi et al., 2024) adopts a learning-theoretic regression framework, and BSACC (Borah et al., 2026) uses cubic B-spline reconstruction. These methods can exploit additional worker responses to improve approximation accuracy. More recently, Moradi et al. (2025) develop a general coded-computing framework and establish vanishing expected reconstruction error under independent random straggling. Their empirical evaluation includes both polynomial computations and deep neural networks. In their polynomial experiments, Moradi et al. (2025) treat LCC as achieving exact recovery once the classical threshold is met. This comparison does not examine the finite-precision conditioning of the polynomial decoder above the recovery threshold.

While the aforementioned challenges highlight the limitations of square decoding matrices for polynomial decoding, they also reveal an untapped opportunity. In a practical distributed system, the master node typically receives $N > D$ responses before the absolute slowest stragglers arrive. For approximation-based schemes such as BACC and BSACC, incorporating these additional responses is an established strategy to reduce approximation error. However, in exact polynomial coded computing, any response acquired after the classical recovery threshold D is traditionally discarded as it is algebraically redundant. This divergence motivates the central questions of this work:

1. When $N > D$ worker responses are available, can this redundancy be exploited to improve the polynomial reconstruction accuracy?
2. Can we establish a more accurate distributed computing scheme for a general function?

To answer the first question, a natural approach is to retain all N responses and solve an overdetermined least-squares problem via Least Square Estimate (LSE) instead of selecting randomly D responses and solving a square linear system. However, the numerical advantage of doing so is not automatic. Let $\boldsymbol{A}_N \in \mathbb{R}^{N\times D}$ denote the decoding matrix. The stability of the overdetermined linear system is governed by the condition number $\kappa(\boldsymbol{A}_N)$ (as detailed in Subsection 2.2). When an additional worker responds, their evaluation forms a new row $\boldsymbol{r}$, updating the Gram matrix to $\boldsymbol{A}_{N+1}^\top \boldsymbol{A}_{N+1} = \boldsymbol{A}_N^\top \boldsymbol{A}_N + \boldsymbol{r}^\top \boldsymbol{r}$. It is well-known that such an update increases both the maximum and minimum eigenvalues. Since both the numerator and the denominator increase, the condition number is not guaranteed to decrease, it may actually increases.

The problem is further complicated by the unpredictable nature of stragglers. In a practical distributed setting, the master node cannot control or predict which specific workers will respond in time. The rows of the decoding matrix are constructed from the evaluation points of these responding servers. Consequently, the decoding matrix is a random matrix. Understanding whether additional responses actually improve numerical stability therefore requires analyzing the extreme eigenvalues of the resulting random Gram matrices.

Furthermore, to answer the second question, we revisit the simple repetition strategy, in which each task is replicated across multiple servers. In coding theory, repetition is generally viewed unfavorably due to its poor erasure correction capabilities, leading to weak straggler resistance in worst-case scenarios. For instance, suppose a distributed system of 100 servers is partitioned into 5 groups of 20 servers, with each group assigned to compute one of 5 distinct tasks. To deterministically guarantee the exact recovery of all

tasks, the master node must wait for 81 responses. However, this worst-case metric is overly pessimistic for practical distributed inference. Under a random straggler model, even with only 30 responding servers, the probability of collecting at least one responsive server from every group remains exceptionally high. Nevertheless, despite this high probability, strict reliance on repetition cannot guarantee an output if a task happens to be completely dropped. To resolve this, we establish a hybrid approach for the non-polynomial components that combines repetition with approximation-based coded computing. Under this design, in the highly probable best-case scenario, we obtain the exact computation of the target tasks, completely bypassing the fundamental difficulties of reconstructing general functions. When exact recovery fails because a task is dropped, the system seamlessly falls back on the available responses to compute an approximation with a theoretically bounded error.

1.1 OUR CONTRIBUTIONS

In this work, we propose two coded distributed computing frameworks, tailored for heterogeneous machine learning architectures, consisting of polynomial and non-polynomial computations. Rather than treating an entire inference model uniformly, we explicitly distinguish between its algebraic structures to improve accuracy while having straggler resistance. Our specific contributions are summarized as follows:

- **Overdetermined Stable Polynomial Decoding:** For polynomial subnetworks, we formulate ChebyLS (Section 3), a decoder that retains all $N > D$ available worker responses instead of discarding responses beyond the classical recovery threshold. We show that adding responses cannot increase the least-squares amplification factor $\|\boldsymbol{R}^{\dagger}\|_2$, and under a random straggler model we derive a probabilistic condition-number bound by combining coherence-based row-sampling inequalities with the deterministic structure of the full Chebyshev-Vandermonde matrix and concentration of the complementary missing set. Our numerical experiments show that this improved conditioning translates into substantially lower finite-precision reconstruction error. We derive a probabilistic condition-number bound that sharpens a general coherence-based row-sampling result by exploiting deterministic structure of the full Chebyshev-Vandermonde matrix and concentration of the missing workers (see Theorem 1).
- **Non-Polynomial Recovery:** For non-polynomial subnetworks, we introduce RepApprox (in Section 4), a hybrid repetition-approximation strategy. We show that the probability of achieving the best-case scenario of exactly recovering all non-polynomial tasks is high under a random straggler model (see Theorem 2). This guarantees exact inference for the vast majority of executions, while allowing the system to fall back on bounded-error approximation in the rare worst-case scenario.
- **Heterogeneous Empirical Evaluation:** We validate the proposed hybrid framework on inference tasks encompassing strictly polynomial models and mixed architectures (such as parallel Deep & Cross Networks) in Section 5. Our experiments confirm that exploiting extra responses in both regimes translates to lower overall relative error compared to some state-of-the-art benchmarks.

2 PRELIMINARIES

2.1 NOTATIONS

For any integer a, we use $[a]$ to denote $\{1, 2, \ldots, a\}$. Unless specified, we use K to denote the number of data, $f(\cdot)$ to denote the computation polynomial function of degree d, $u(\cdot)$ to denote encoding polynomial, $h(\cdot) = f(u(\cdot))$ to denote the target polynomial to decode of degree $D - 1$. We consider a distributed system with M servers and s of them are stragglers, and we let $N = M - s$ be the number of responding servers. We use β_i to denote the evaluation points assigned to data $\boldsymbol{X}_i$ and γ_i to denote evaluation points assigned to server i. We use $\{\alpha_i\}_{i=1}^{N} \subseteq \{\gamma_i\}_{i=1}^{M}$ as the set of evaluation points of N responding servers.

We use bold lower-case $\boldsymbol{x}$ and upper-case $\boldsymbol{X}$ to denote a vector and matrix, respectively. For a vector $\boldsymbol{x}$, $||\boldsymbol{x}||_2$ denotes the Euclidean norm of $\boldsymbol{x}$. For a matrix $\boldsymbol{A}$, $||\boldsymbol{A}||_{\text{op}}$ denotes the operator norm of $\boldsymbol{A}$ defined as $||\boldsymbol{A}||_{\text{op}} = \sup_{||\boldsymbol{v}||_2=1} ||\boldsymbol{A}\boldsymbol{v}||_2$ and $||\boldsymbol{A}||_F$ denotes the Frobenius norm. For a symmetric positive semi-definite matrix $\boldsymbol{M}$, we denote its largest and smallest eigenvalues as $\lambda_{\max}(\boldsymbol{M})$ and $\lambda_{\min}(\boldsymbol{M})$, respectively.

2.2 Condition Number of an Overdetermined Linear System

To simplify the notation, we let the exact response from server i be vectorized into a row vector $\boldsymbol{y}_i \in \mathbb{R}^{1\times p}$, where p is the total number of elements in the response. We stack the N server responses to form the response matrix $\boldsymbol{Y} \in \mathbb{R}^{N\times p}$ and the D unknown polynomial coefficients into a coefficient matrix $\boldsymbol{C} \in \mathbb{R}^{D\times p}$.

Consider an overdetermined linear matrix equation $\boldsymbol{A}\boldsymbol{C} = \boldsymbol{Y}$, where $\boldsymbol{A} \in \mathbb{R}^{N\times D}$ is the decoding matrix. Under exact arithmetic, the parameters can be recovered exactly by choosing only D responses out of N responses $\boldsymbol{Y}$ for the linear system and adjust the matrix $\boldsymbol{A}$ accordingly. Furthermore, although redundant, under exact arithmetic, we can also use all $N > D$ responses and obtain $\boldsymbol{C}$ via Least Squares Estimation (LSE). That is, let $\boldsymbol{A}^\dagger \triangleq (\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top$, then $\boldsymbol{C} = \boldsymbol{A}^\dagger\boldsymbol{Y}$. However, because operations are performed under finite-precision arithmetic, server i unintentionally returns a perturbed response $\hat{\boldsymbol{y}}_i = \boldsymbol{y}_i + \boldsymbol{\Delta y}_i$. Consequently, the master node observes the corrupted matrix $\hat{\boldsymbol{Y}} = \boldsymbol{Y} + \boldsymbol{\Delta Y}$. The corrupted solution $\hat{\boldsymbol{C}}$ therefore satisfies $\hat{\boldsymbol{C}} = \boldsymbol{A}^\dagger\hat{\boldsymbol{Y}}$.

By substituting $\hat{\boldsymbol{Y}}$ and isolating the error, the deviation of the recovered coefficients is given by $\Delta\boldsymbol{C} = \boldsymbol{A}^\dagger\Delta\boldsymbol{Y}$. Taking the Frobenius norm ($||\cdot||_F$) on both sides and applying sub-multiplicativity with the operator norm ($||\cdot||_{\text{op}}$), we can bound the recovery relative error as

$$\frac{||\Delta\boldsymbol{C}||_F}{||\boldsymbol{C}||_F} \le \kappa(\boldsymbol{A}) \cdot \frac{||\Delta\boldsymbol{Y}||_F}{||\boldsymbol{Y}||_F}, \quad \text{with} \quad \kappa(\boldsymbol{A}) = ||\boldsymbol{A}||_{\text{op}}||\boldsymbol{A}^\dagger||_{\text{op}} = \sqrt{\frac{\lambda_{\max}(\boldsymbol{A}^\top\boldsymbol{A})}{\lambda_{\min}(\boldsymbol{A}^\top\boldsymbol{A})}}.$$

We call $\kappa(\boldsymbol{A})$ the *condition number* of the linear system.

2.3 Previous Works on Polynomial Coded Computing

Lagrange Coded Computing (LCC) (Yu et al., 2019) establishes the foundational architecture for polynomial distributed computing. The master node constructs an encoding polynomial $u(x)$ that interpolates K data, and asks workers to evaluate $h(x) = f(u(x))$. However, LCC operates over finite fields. Investigating its direct application to the continuous real-number domain, Soleymani et al. (2021) proposed Analog Lagrange Coded Computing (ALCC). In that work, ALCC addressed numerical stability in the decoding phase and lowered the condition number of the system by choosing complex roots of unity as evaluation points for both data and servers. However, this introduces significant computational overhead. By contrast, constructing a standard Vandermonde matrix with uniformly spaced evaluation points on the real line yields an exponentially high condition number (see Figure 1).

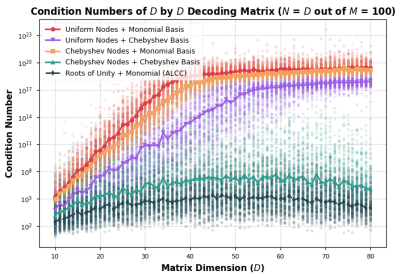

Figure 1: Condition number of D by D matrix across different choices of evaluation points and polynomial bases.

Fahim & Cadambe (2021) proposed a numerically stable coded distributed framework for matrix multiplication and also a numerically stable LCC scheme over real domain. To mitigate numerical instability, they proposed the use of Chebyshev nodes for evaluation points and expressed the target computation polynomial $h(x)$ of degree $D-1$ over Chebyshev polynomial basis. Suppose that the image of the target computation is a vector, we let $\boldsymbol{c}_j \in \mathbb{R}^{1\times p}$ denote the j-th polynomial coefficient row vector. Thus, $h(x)$ is defined as

$h(x) = \boldsymbol{c}_0 \frac{T_0(x)}{\sqrt{2}} + \sum_{j=1}^{D-1} \boldsymbol{c}_j T_j(x)$, where $T_j(x)$ is the Chebyshev polynomial of the first kind of degree j, defined recursively as $T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$. The master node sets the data and server evaluation points to be Chebyshev nodes $\beta_i = \cos(\frac{(2i-1)\pi}{2k})$ for $i \in [k]$ and $\gamma_i = \cos(\frac{(2i-1)\pi}{2M})$ for $i \in [M]$, respectively. Let $\{\alpha_i\}_{i=1}^N \subseteq \{\gamma_i\}_{i=1}^M$ be the evaluation points of the responding servers. By stacking the coefficient vectors $\boldsymbol{c}_j$ to construct the global coefficient matrix $\boldsymbol{C} \in \mathbb{R}^{D\times p}$ and the exact server responses $h(\alpha_i)$ to form the observation matrix $\boldsymbol{Y} \in \mathbb{R}^{N\times p}$, the decoding phase requires solving the linear system $\boldsymbol{RC} = \boldsymbol{Y}$. The decoding matrix $\boldsymbol{R} \in \mathbb{R}^{N\times D}$ is defined as:

$$\boldsymbol{R} = \begin{bmatrix} T_0(\alpha_1)/\sqrt{2} & T_1(\alpha_1) & \dots & T_{D-1}(\alpha_1) \\ T_0(\alpha_2)/\sqrt{2} & T_1(\alpha_2) & \dots & T_{D-1}(\alpha_2) \\ \vdots & \vdots & \ddots & \vdots \\ T_0(\alpha_N)/\sqrt{2} & T_1(\alpha_N) & \dots & T_{D-1}(\alpha_N) \end{bmatrix}.$$

Fahim & Cadambe (2021) uses exactly $N = D$ server responses, i.e., $\boldsymbol{R}$ is a square Chebyshev-Vandermonde matrix. Despite the efforts, the condition number of a square Chebyshev Vandermonde matrix $\boldsymbol{R}$ with Chebyshev nodes is high (see Figure 1). To address this limitation, we introduce ChebyLS (described in Section 3), a decoding strategy that incorporates $N > D$ responses to further reduce the condition number and stabilize the polynomial reconstruction.

2.4 Previous Works on Approximation Based Coded Computing

To address the computation of non-polynomial functions used in neural networks (such as ReLUs and Softmax), recent frameworks abandon the polynomial assumption. While this sacrifices strict exact recovery, these methods achieve a theoretically bounded approximation error. Specifically, Jahani-Nezhad & Maddah-Ali (2023) introduced Berrut Approximated Coded Computing (BACC), which evaluates a general function $f(x)$. The data is encoded with a rational function $v(x)$ via Barycentric interpolation. Then, the master node approximates the target computation by interpolating the N available server responses $\{f(v(\alpha_i))\}_{i\in[N]}$ with a rational function $r(x)$, again via Barycentric interpolation. The accuracy of this approach is measured by the approximation error $||r(x) - f(v(x))||$, which is theoretically bounded. Building upon this paradigm, Borah et al. (2026) proposed Basis Spline Assisted Coded Computing (BSACC), which employs natural cubic B-splines for approximation, improving the approximation error bound.

While approximation-based schemes achieve a bounded approximation error, they forfeit the possibility of exact recovery algebraically. To address this limitation, we introduce RepApprox (described in Section 4), a hybrid framework that obtains the exact computation with high probability, while falling back on bounded-error approximation in worst-case scenarios.

3 ChebyLS: Stable Polynomial Coded Distributed Computing

We first address the polynomial regime and formulate the Chebyshev Least Squares (ChebyLS) decoder. By retaining all $N > D$ available worker responses, we replace the square polynomial-decoding system at the classical recovery threshold by the overdetermined system $\boldsymbol{RC} = \boldsymbol{Y}$. The master recovers the coefficient matrix through least squares, $\widehat{\boldsymbol{C}} = \boldsymbol{R}^{\dagger}\boldsymbol{Y}$. As discussed in Appendix 2.2, the sensitivity of this recovery to perturbations in the worker responses is governed by the singular values of $\boldsymbol{R}$.

There is first a simple deterministic benefit to retaining additional responses. If $\boldsymbol{R}_{N+1}$ is obtained from $\boldsymbol{R}_N$ by appending one additional worker response, then $\boldsymbol{R}_{N+1}^{\top}\boldsymbol{R}_{N+1} \succeq \boldsymbol{R}_N^{\top}\boldsymbol{R}_N$. Consequently, $\sigma_{\min}(\boldsymbol{R}_{N+1}) \geq$

$\sigma_{\min}(\boldsymbol{R}_N)$ and hence $\|\boldsymbol{R}_{N+1}^{\dagger}\|_2 \le \|\boldsymbol{R}_N^{\dagger}\|_2$ (Proposition 3). Thus, adding worker responses cannot increase the worst-case least-squares amplification factor associated with the pseudoinverse.

This monotonicity does not imply that the condition number $\kappa(\boldsymbol{R}_N)$ itself decreases for every sequence of arriving workers, since both the largest and smallest singular values may increase. Moreover, under unpredictable worker latency, the rows available to the master form a random subset of the complete Chebyshev decoding matrix. We therefore study the extreme eigenvalues of the random Gram matrix $\boldsymbol{R}^{\top}\boldsymbol{R}$ under a random straggler model.

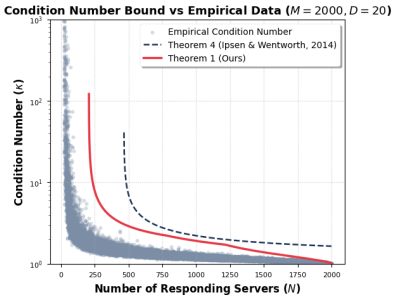

Figure 2: Condition-number upper bounds for the ChebyLS decoder ($M = 2000$, $D = 20$, $\delta = 0.1$). The empirical condition numbers lie below the bound in Theorem 1 and decrease as additional worker responses become available. The ChebyLS-specific bound also improves upon the general coherence bound of Ipsen & Wentworth (2014, Theorem 4.1).

To obtain a condition-number guarantee, we normalize the complete Chebyshev decoding matrix to an orthonormal-column matrix and apply the coherence-based row-sampling bounds of Ipsen & Wentworth (2014). We sharpen the general bound by exploiting two additional features of the ChebyLS construction: deterministic information from the complete Chebyshev frame, and concentration of the complementary set of missing workers. Combining these bounds yields the following result. Detailed derivations are provided in Appendix A.

Theorem 1 (ChebyLS condition-number bound). *Fix $\delta \in (0, 1/2)$ and let $f(x) \triangleq e^x/(1+x)^{1+x}$ for $x > -1$. For $n > 0$, let $\epsilon_+(n,\delta) > 0$ be the unique solution of $Df(\epsilon_+(n,\delta))^{n/(2D-1)} = \delta$, and define $\mathsf{U}_n(\delta) \triangleq \frac{n}{2}(1+\epsilon_+(n,\delta))$. Similarly, whenever a solution $\epsilon_-(n,\delta) \in (0,1)$ exists, let it satisfy $Df(-\epsilon_-(n,\delta))^{n/(2D-1)} = \delta$, and define $\mathsf{L}_n(\delta) \triangleq \frac{n}{2}(1-\epsilon_-(n,\delta))$. If the lower-tail equation has no solution, set $\mathsf{L}_n(\delta) = 0$. We also set $\mathsf{U}_0(\delta) = \mathsf{L}_0(\delta) = 0$. Let $s = M - N$. Then, with probability at least $1 - 2\delta$,*

$$\kappa(\boldsymbol{R}) \le \sqrt{\frac{\min\left\{\frac{M}{2},\, \mathsf{U}_N(\delta),\, \frac{M}{2} - \mathsf{L}_s(\delta)\right\}}{\max\left\{\frac{M}{2}\left[1 - \frac{s(2D-1)}{M}\right]_+,\, \mathsf{L}_N(\delta),\, \left[\frac{M}{2} - \mathsf{U}_s(\delta)\right]_+\right\}}},$$

provided that the denominator is positive, where $[x]_+ \triangleq \max\{x, 0\}$.

Theorem 1 quantifies the conditioning of ChebyLS under a random set of responses. Figure 2 shows that the resulting bound decreases substantially as additional responses become available and is sharper than the general coherence bound of Ipsen & Wentworth (2014, Theorem 4.1), particularly when N is close to M. Consequently, retaining algebraically redundant responses suppresses the amplification of finite-precision noise, resulting in a more numerically stable polynomial reconstruction.

4 RepApprox: Improving Approximation-Based Coded Computing

As discussed in Section 2.4, while approximation-based schemes achieve a bounded approximation error for general functions, they forfeit the exact recovery algebraically. To bridge this gap, we introduce RepApprox, a hybrid repetition-approximation coded computing framework. Although RepApprox is a generalized framework applicable to any distributed computation, it is explicitly designed to bypass the fundamental reconstruction bottlenecks associated with non-polynomial functions.

RepApprox works as follows. Suppose we have K tasks $\{f(\boldsymbol{x})\}_{i\in[K]}$, and each task is assigned to c distinct servers, forming a repetition group of size $M_{\text{rep}} = cK \le M$. Specifically, the master node forms K groups

$G_1, \ldots, G_K$, $|G_i| = c$, such that each servers in G_i is computing $f(\boldsymbol{x}_i)$. Simultaneously, the remaining $M - cK$ servers are utilized to perform approximation-based coded computing. Consequently, the responses from the repetition servers take the form

$$\{\underbrace{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_1)}_{\text{from } G_1}, \underbrace{f(\boldsymbol{x}_2), \ldots, f(\boldsymbol{x}_2)}_{\text{from } G_2}, \ldots, \underbrace{f(\boldsymbol{x}_K), \ldots, f(\boldsymbol{x}_K)}_{\text{from } G_K}\}.$$

The master node then waits for N responses to arrive from M servers. Note that the master can algebraically recover all K tasks exactly, provided there is at least one response from every group G_i. Theorem 2 demonstrates that under a random straggler model, this scenario is highly probable. We provide its proof in Appendix B.

Theorem 2. *Consider a distributed computing system with M total servers and K tasks, each replicated c times such that $cK \leq M$. Assuming the master node receives N responses drawn uniformly at random from M servers, the probability of exactly recovering all K tasks is*

$$p_{exact} = \sum_{j=0}^{K} (-1)^j \binom{K}{j} \frac{\binom{M-jc}{N}}{\binom{M}{N}} \geq 1 - K\left(1 - N/M\right)^c.$$

Theorem 2 shows that increasing the replication factor c increases the success probability. To maximize the guarantee of exact recovery, one might naturally configure c to its maximum allowable value, $c = \lfloor M/K \rfloor$. However, doing so dedicates almost all available servers to exact repetitions, leaving few, if any, servers in the remaining $M - cK$ pool available to perform threshold-free approximation-based coded computing.

In the exceptionally rare worst-case scenario where every server in a specific group G_i becomes a straggler, the exact result $f(\boldsymbol{x}_i)$ is irreversibly lost. Although this condition is statistically highly improbable, a robust distributed framework must prevent the total inability to obtain a specific computation result at all. Rather than setting the replication factor c arbitrarily, Theorem 2 provides an operational guideline for system design. Suppose the system requires the exact recovery to succeed with a target probability of at least p, given a deadline threshold of responses collected N. That is, $c \geq \ln\left(\frac{1-p}{K}\right) / \ln\left(1 - \frac{N}{M}\right)$. To illustrate, let us consider the AWS Lambda distributed environment (Gupta et al., 2018) where tail latencies typically bottleneck the slowest 5% of workers. Assuming the master node waits for the 95th percentile of responses ($N/M = 0.95$), and suppose $K = 10$, the system designer may target an exact recovery probability of at least $p = 0.90$. Applying the formulation yields $c \geq \ln(0.01)/\ln(0.05) \approx 1.53$. Hence with $c = 2$, the system secures a 97.5% probability of exact recovery using a repetition group of only $M_{rep} = 20$ servers. The remaining $M - 20$ servers in the cluster are subsequently dedicated to performing approximation-based coded computing.

5 Numerical Experiment

In this section, we evaluate the numerical stability and recovery accuracy of our proposed coded distributed computing scheme on the Deep & Cross Network v2 (DCN v2) architecture (Wang et al., 2021). DCN v2 consists of two sub-networks: it processes embedded inputs simultaneously through a strictly polynomial Cross Network and a non-polynomial Deep Network. Because of this structural dichotomy, it serves as an ideal model for evaluating exact and approximation-based coded computing paradigms.

We evaluate our proposed coding frameworks using the Deep & Cross Network v2 (DCN v2) architecture (Wang et al., 2021) on the MovieLens 100K dataset, embedding each user and item into a 16-dimensional feature space. To systematically evaluate our contributions, we consider two cases:

- **Cross-Only Network:** We evaluate the relative recovery error of our overdetermined **ChebyLS** decoder against four distinct baselines: Threshold Chebyshev (Fahim & Cadambe, 2021) and

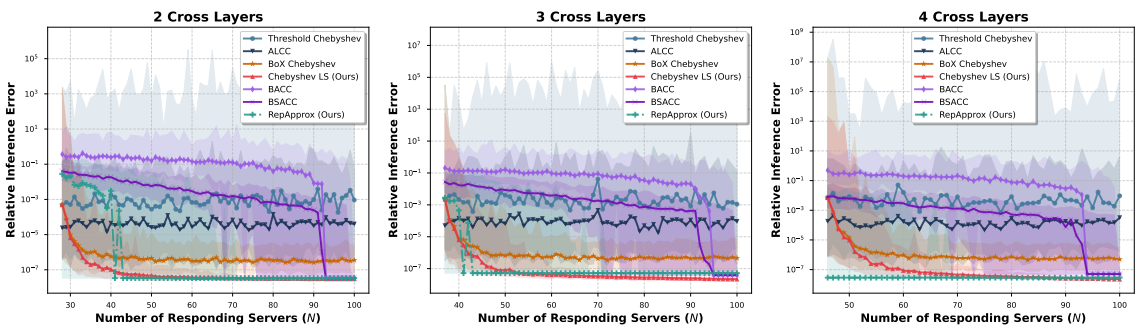

Figure 3: End-to end relative inference error for Cross-Only architectures ($K = 10$, $M = 100$).

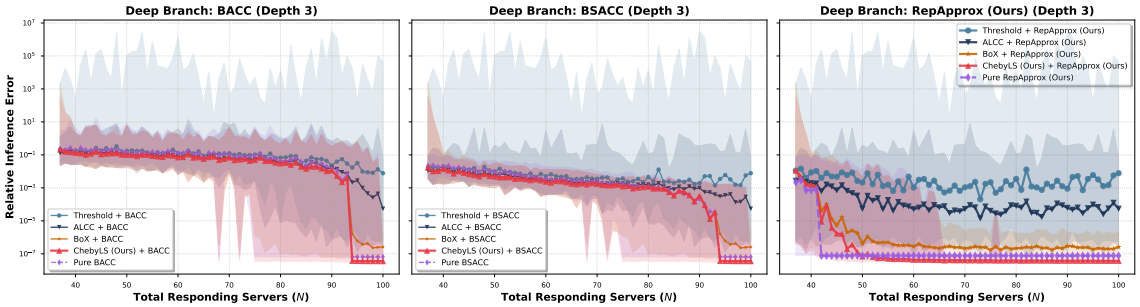

Figure 4: End-to-end relative inference error for the Parallel DCN v2 with depth 3 ($K = 10$, $M = 100$).

ALCC (Soleymani et al., 2021), which solves a square linear system from a random subset of exactly D responses; Best-of-X (BoX) Chebyshev, which decodes using the best-conditioned square Chebyshev-Vandermonde subsystem from X randomly sampled subsets[1]; BACC (Jahani-Nezhad & Maddah-Ali, 2023), which approximates the target function using barycentric rational interpolation; and BSACC (Borah et al., 2026), which approximates the computation via basis spline interpolation. We also compare with our RepApprox.

- **Parallel DCN v2:** To rigorously isolate the performance gains of our contributions, we compare against an extensive suite of baselines divided into two categories: (1) *Alternative Hybrid Schemes*, which pair exact polynomial decoders (Threshold vs. ALCC vs. BoX vs. ChebyLS) with varying deep branch decoders (BACC, BSACC, or RepApprox), and (2) *Uniform Schemes*, which treat the entire network as non-polynomial by applying purely BACC, purely BSACC, or purely RepApprox end-to-end.

Due to the space constraint, we defer the details on the experiments, further discussion and additional empirical evaluations to Appendix C and D.

[1]BoX samples X subsets S_j of size D from N available responses and selects $S^* = \arg\min_{j\in[X]} \kappa(\boldsymbol{R}_{S_j})$. BoX incurs substantial additional decoding cost from evaluating X candidate subsets, and is therefore used as a benchmark rather than a practical decoding strategy. Fro experiments in Figure 3 and 4, we use $X = 200$. We further analyze the condition number of various X (see Figure 5)

REFERENCES

Rimpi Borah, J. Harshan, and V. Lalitha. Basis-spline assisted coded computing: Strategies and error bounds. In *2026 IEEE International Symposium on Information Theory (ISIT)*, pp. 1–6, 2026. doi: 10.1109/ISIT62367.2026.11653858.

Grigorios G. Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Yannis Panagakis, Jiankang Deng, and Stefanos Zafeiriou. P–nets: Deep polynomial neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7323–7333, 2020. doi: 10.1109/CVPR42600.2020.00735.

Anindya Bijoy Das, Aditya Ramamoorthy, and Namrata Vaswani. Efficient and robust distributed matrix computations via convolutional coding. *IEEE Transactions on Information Theory*, 67(9):6266–6282, 2021.

Jeffrey Dean and Luiz André Barroso. The tail at scale. *Commun. ACM*, 56(2):74–80, February 2013. ISSN 0001-0782. doi: 10.1145/2408776.2408794. URL https://doi.org/10.1145/2408776.2408794.

Sanghamitra Dutta, Mohammad Fahim, Farzin Haddadpour, Haewon Jeong, Viveck Cadambe, and Pulkit Grover. On the optimal recovery threshold of coded matrix multiplication. *IEEE Transactions on Information Theory*, 66(1):278–301, 2020. doi: 10.1109/TIT.2019.2929328.

Mohammad Fahim and Viveck R. Cadambe. Numerically stable polynomially coded computing. *IEEE Transactions on Information Theory*, 67(5):2758–2785, 2021. doi: 10.1109/TIT.2021.3050526.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*. MIT press Cambridge, 2016.

Vipul Gupta, Shusen Wang, Thomas Courtade, and Kannan Ramchandran. Oversketch: Approximate matrix multiplication for the cloud. In *2018 IEEE International Conference on Big Data (Big Data)*, pp. 298–304, 2018. doi: 10.1109/BigData.2018.8622139.

Ilse C. F. Ipsen and Thomas Wentworth. The effect of coherence on sampling from matrices with orthonormal columns, and preconditioned least squares problems. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1490–1520, 2014. doi: 10.1137/120870748.

Tayyebeh Jahani-Nezhad and Mohammad Ali Maddah-Ali. Berrut approximated coded computing: Straggler resistance beyond polynomial computing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):111–122, 2023. doi: 10.1109/TPAMI.2022.3151434.

Wilton Kim, Stanislav Kruglik, and Han Mao Kiah. Verifiable coded computation of multiple functions. *IEEE Transactions on Information Forensics and Security*, 19:8009–8022, 2024. doi: 10.1109/TIFS.2024.3450288.

Songze Li and Salman Avestimehr. Coded computing: Mitigating fundamental bottlenecks in large-scale distributed computing and machine learning. *Found. Trends Commun. Inf. Theory*, 17(1):1–148, August 2020. ISSN 1567-2190. doi: 10.1561/0100000103. URL https://doi.org/10.1561/0100000103.

Parsa Moradi, Behrooz Tahmasebi, and Mohammad A Maddah-Ali. Coded computing for resilient distributed computing: A learning-theoretic framework. *Advances in Neural Information Processing Systems*, 37:111923–111964, 2024.

Parsa Moradi, Hanzaleh Akbarinodehi, and Mohammad Ali Maddah-Ali. General coded computing: Adversarial settings. In *2025 IEEE International Symposium on Information Theory (ISIT)*, pp. 1–6, 2025. doi: 10.1109/ISIT63088.2025.11195419.

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

Adarsh M. Subramaniam, Anoosheh Heidarzadeh, Asit Kumar Pradhan, and Krishna R. Narayanan. Product lagrange coded computing. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pp. 197–202, 2020. doi: 10.1109/ISIT44484.2020.9174440.

Li Tang, Konstantinos Konstantinidis, and Aditya Ramamoorthy. Erasure coding for distributed matrix multiplication for matrices with bounded entries. *IEEE Communications Letters*, 23(1):8–11, 2018.

Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, ADKDD'17, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450351942. doi: 10.1145/3124749.3124754. URL https://doi.org/10.1145/3124749.3124754.

Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*, WWW '21, pp. 1785–1797, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3450078. URL https://doi.org/10.1145/3442381.3450078.

Qian Yu, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. Polynomial codes: an optimal design for high-dimensional coded matrix multiplication. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 4406–4416, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

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

Jinbao Zhu and Songze Li. Generalized lagrange coded computing: A flexible computation-communication tradeoff. In *2022 IEEE International Symposium on Information Theory (ISIT)*, pp. 832–837. IEEE, 2022.

A CONDITION NUMBER OF CHEBYLS DECODING MATRIX

In this section, we study the conditioning of the ChebyLS decoding matrix under a random straggler model. We first establish a monotonicity property of least-squares recovery as additional worker responses become available. We then normalize the complete Chebyshev decoding matrix to obtain a matrix with orthonormal columns, placing ChebyLS within the coherence-based row-sampling framework of Ipsen & Wentworth (2014).

The general coherence bound of Ipsen & Wentworth (2014) controls the condition number of a uniformly sampled row submatrix using concentration inequalities. For the ChebyLS construction, we can sharpen this result by exploiting additional structure. First, the complete Chebyshev decoding matrix provides deterministic bounds on the extreme eigenvalues of every responder submatrix. Second, since the missing workers form the complementary uniformly random subset, the same concentration inequalities can be applied to the missing rows. Through the identity $\boldsymbol{Q}_{\mathcal{S}}^{\top}\boldsymbol{Q}_{\mathcal{S}} = \boldsymbol{I}_D - \boldsymbol{Q}_{\mathcal{T}}^{\top}\boldsymbol{Q}_{\mathcal{T}}$, these missing-set bounds yield additional bounds on the extreme eigenvalues of the responder Gram matrix. Combining the strongest available upper and lower bounds gives Theorem 1.

A.1 MONOTONICITY OF CHEBYLS RECOVERY

We begin with a property that does not require random stragglers, Chebyshev nodes, or any distributional assumption. Suppose that $\boldsymbol{R}_N$ is a full column-rank decoding matrix formed from $N \geq D$ responses, and that $\boldsymbol{R}_{N+1}$ is obtained by appending one additional response row $\boldsymbol{r}^{\top}$. Then adding this response contributes a positive semidefinite rank-one matrix to the Gram matrix.

Proposition 3 (Monotonicity with additional responses)**.** *For any nested sequence of responder sets with $N \geq D$,*

$$\begin{aligned}
&\boldsymbol{R}_{N+1}^{\top}\boldsymbol{R}_{N+1} = \boldsymbol{R}_N^{\top}\boldsymbol{R}_N + \boldsymbol{r}\boldsymbol{r}^{\top} \succeq \boldsymbol{R}_N^{\top}\boldsymbol{R}_N, \\
&\sigma_{\min}(\boldsymbol{R}_{N+1}) \geq \sigma_{\min}(\boldsymbol{R}_N), \qquad \|\boldsymbol{R}_{N+1}^{\dagger}\|_2 \leq \|\boldsymbol{R}_N^{\dagger}\|_2.
\end{aligned}$$

Proof. The first relation follows because $\boldsymbol{r}\boldsymbol{r}^{\top} \succeq 0$. Hence the minimum eigenvalue of the Gram matrix cannot decrease when a row is appended. Since $\|\boldsymbol{R}_N^{\dagger}\|_2 = 1/\sigma_{\min}(\boldsymbol{R}_N)$ for a full column-rank matrix, the final inequality follows immediately. □

Thus, additional responses cannot increase the worst-case amplification factor $\|\boldsymbol{R}_N^{\dagger}\|_2$ for an additive perturbation in the worker responses. Notice that this statement concerns the smallest singular value, rather than the condition number itself. In general, $\kappa(\boldsymbol{R}_N)$ need not decrease monotonically with N.

A related monotonicity statement holds under a standard stochastic perturbation model. For one output coordinate, suppose $\boldsymbol{y}_N = \boldsymbol{R}_N\boldsymbol{c} + \boldsymbol{\varepsilon}_N$, where the response errors are zero mean and satisfy $\mathbb{E}[\boldsymbol{\varepsilon}_N\boldsymbol{\varepsilon}_N^{\top}] = \sigma^2\boldsymbol{I}$. If $\boldsymbol{B}$ evaluates the recovered polynomial at the desired input points, then

$$\mathbb{E}\big[\|\boldsymbol{B}(\widehat{\boldsymbol{c}}_{N+1} - \boldsymbol{c})\|_2^2\big] \leq \mathbb{E}\big[\|\boldsymbol{B}(\widehat{\boldsymbol{c}}_N - \boldsymbol{c})\|_2^2\big] = \sigma^2 \operatorname{Tr}\big(\boldsymbol{B}(\boldsymbol{R}_N^{\top}\boldsymbol{R}_N)^{-1}\boldsymbol{B}^{\top}\big)\,.$$

This stochastic model provides a direct connection between additional responses and reconstruction accuracy. We emphasize that it is a stylized additive-noise model and is not intended to model floating-point roundoff exactly.

A.2 Connection to coherence-based row sampling

Let $\boldsymbol{R}_{\text{full}} \in \mathbb{R}^{M\times D}$ denote the decoding matrix when all M workers respond. We normalize the complete matrix by defining $\boldsymbol{Q} = \sqrt{2/M}\,\boldsymbol{R}_{\text{full}}$. The discrete orthogonality of the Chebyshev basis at the first-kind Chebyshev nodes implies that $\boldsymbol{Q}$ has orthonormal columns. Let $\boldsymbol{q}_j^\top$ denote its j-th row and define its coherence by $\mu = \max_{j\in[M]} \|\boldsymbol{q}_j\|_2^2$. Since $|T_k(x)| \le 1$ on $[-1,1]$, the Chebyshev construction also gives the simple coherence estimate

$$\boldsymbol{Q}^\top\boldsymbol{Q} = \boldsymbol{I}_D, \qquad \mu = \max_{j\in[M]} \|\boldsymbol{q}_j\|_2^2 \le \frac{2D-1}{M}.$$

This places ChebyLS directly in the framework studied by Ipsen & Wentworth (2014). Their main coherence result applies to uniform sampling without replacement, uniform sampling with replacement, and Bernoulli sampling. We state below the without-replacement specialization relevant to our setting. Let $f(x) = e^x/(1+x)^{1+x}$.

Theorem 4 (General coherence bound (Ipsen & Wentworth, 2014, Theorem 4.1))**.** *Let $\boldsymbol{Q} \in \mathbb{R}^{M\times D}$ satisfy $\boldsymbol{Q}^\top\boldsymbol{Q} = \boldsymbol{I}_D$ and have coherence μ. Let $\mathcal{S}$ be a uniformly random subset of N rows, where $D \le N \le M$. For $0 < \epsilon < 1$, define δ_{IW} as below. If $\delta_{\text{IW}} < 1$, then with probability at least $1 - \delta_{\text{IW}}$, $\boldsymbol{Q}_{\mathcal{S}}$ has full column rank and*

$$\delta_{\text{IW}} = D\left[f(-\epsilon)^{N/(M\mu)} + f(\epsilon)^{N/(M\mu)}\right], \qquad \kappa(\boldsymbol{Q}_{\mathcal{S}}) \le \sqrt{\frac{1+\epsilon}{1-\epsilon}}.$$

Theorem 4 is the result stated explicitly by Ipsen & Wentworth (2014). For our purposes, it is useful to retain the upper and lower eigenvalue estimates separately, since they can later be combined with deterministic information.

The following two inequalities are the corresponding tail estimates used in Appendix A.2 of Ipsen & Wentworth (2014) to prove their Theorem 4.1. They are not presented there as separate named theorems. We state them separately here because this allows the upper and lower tails to be optimized independently.

Since $\boldsymbol{R}^\top\boldsymbol{R} = (M/2)\boldsymbol{Q}_{\mathcal{S}}^\top\boldsymbol{Q}_{\mathcal{S}}$, their upper tail gives the following result.

Theorem 5 (Responder upper-tail bound)**.** *For every $\epsilon > 0$,*

$$\Pr\left\{\lambda_{\max}(\boldsymbol{R}^\top\boldsymbol{R}) \ge \frac{N}{2}(1+\epsilon)\right\} \le D f(\epsilon)^{N/(M\mu)}.$$

For a prescribed per-tail failure probability δ, let ϵ_+^{resp} be the smallest positive value for which the right-hand side is at most δ, and define $P_{\text{high}}^{\text{resp}} = \frac{N}{2}(1+\epsilon_+^{\text{resp}})$. This is a one-dimensional scalar inversion and can be evaluated numerically.

Similarly, the lower tail gives the following result.

Theorem 6 (Responder lower-tail bound)**.** *For every $0 < \epsilon < 1$,*

$$\Pr\left\{\lambda_{\min}(\boldsymbol{R}^\top\boldsymbol{R}) \le \frac{N}{2}(1-\epsilon)\right\} \le D f(-\epsilon)^{N/(M\mu)}.$$

Let $\epsilon_-^{\mathrm{resp}}$ be the smallest value in $(0,1)$ for which the right-hand side is at most δ, whenever such a value exists, and define $P_{\mathrm{low}}^{\mathrm{resp}} = \frac{N}{2}(1-\epsilon_-^{\mathrm{resp}})$. If no such value exists, this probabilistic lower bound is simply non-informative and is omitted.

A.3 Deterministic and missing-response bounds

We next use information that is specific to sampling rows without replacement. Let $\mathcal{T} = [M] \setminus \mathcal{S}$ denote the $s = M - N$ missing workers. Since $\boldsymbol{Q}$ has orthonormal columns, the Gram matrices of the responding and missing rows are complementary: $\boldsymbol{Q}_{\mathcal{S}}^\top \boldsymbol{Q}_{\mathcal{S}} = \boldsymbol{I}_D - \boldsymbol{Q}_{\mathcal{T}}^\top \boldsymbol{Q}_{\mathcal{T}}$. This immediately yields bounds that hold for every choice of the responder set.

Lemma 7 (Deterministic complement bounds)**.** *For every responder set $\mathcal{S}$,*

$$D_{\mathrm{high}} \triangleq \frac{M}{2}, \qquad \lambda_{\max}(\boldsymbol{R}^\top \boldsymbol{R}) \le D_{\mathrm{high}}, \qquad \lambda_{\min}(\boldsymbol{R}^\top \boldsymbol{R}) \ge D_{\mathrm{low}} \triangleq \frac{M}{2}\max\{0, 1-(M-N)\mu\}.$$

Proof. The upper bound follows from $\boldsymbol{Q}_{\mathcal{S}}^\top \boldsymbol{Q}_{\mathcal{S}} \preceq \boldsymbol{I}_D$. For the lower bound, the largest eigenvalue of $\boldsymbol{Q}_{\mathcal{T}}^\top \boldsymbol{Q}_{\mathcal{T}}$ is at most its trace, which is at most $(M-N)\mu$. Substituting this into the complementary Gram-matrix identity proves the result. Using $\mu \le (2D-1)/M$ recovers the deterministic term appearing in Theorem 1. □

There is one further consequence of the complement identity. Since a uniformly random responder set has a uniformly random complement, the Ipsen–Wentworth bounds may also be applied to the $s = M-N$ missing rows. Let $\epsilon_+^{\mathrm{miss}}$ and $\epsilon_-^{\mathrm{miss}}$ denote the corresponding upper- and lower-tail parameters at failure probability δ. Translating those bounds back through the complement identity gives

$$P_{\mathrm{low}}^{\mathrm{miss}} \triangleq \max\left\{0, \frac{M}{2} - \frac{M-N}{2}(1+\epsilon_+^{\mathrm{miss}})\right\}, \qquad P_{\mathrm{high}}^{\mathrm{miss}} \triangleq \frac{M}{2} - \frac{M-N}{2}(1-\epsilon_-^{\mathrm{miss}}).$$

The first is a lower bound for $\lambda_{\min}(\boldsymbol{R}^\top \boldsymbol{R})$ and the second is an upper bound for $\lambda_{\max}(\boldsymbol{R}^\top \boldsymbol{R})$, each with probability at least $1-\delta$ whenever the corresponding Ipsen–Wentworth tail bound is informative.

A.4 Combined condition-number bound

We can now combine the deterministic bound, the concentration bounds for the responding workers, and the corresponding concentration bounds for the missing workers. For fixed M, N, D, μ, and δ, all candidate bounds are deterministic quantities. Thus, before observing the random responder set, we may select whichever candidate gives the smallest upper bound and whichever gives the largest lower bound. Only one probabilistic event is therefore required on each side.

Theorem 8 (Combined coherence bound for ChebyLS)**.** *Provided that the denominator below is positive, with probability at least $1-2\delta$,*

$$\kappa(\boldsymbol{R}) \le \sqrt{\frac{\min\left\{D_{\mathrm{high}}, P_{\mathrm{high}}^{\mathrm{resp}}, P_{\mathrm{high}}^{\mathrm{miss}}\right\}}{\max\left\{D_{\mathrm{low}}, P_{\mathrm{low}}^{\mathrm{resp}}, P_{\mathrm{low}}^{\mathrm{miss}}\right\}}}.$$

Proof. The deterministic bounds hold for every responder set. Among the two probabilistic candidates for the upper eigenvalue, choose in advance the one giving the smaller numerical bound. Its failure probability is

at most δ. Likewise, choose in advance the larger of the two probabilistic lower-eigenvalue bounds; its failure probability is also at most δ. A union bound therefore shows that the selected upper and lower inequalities hold simultaneously with probability at least $1 - 2\delta$. Taking the ratio of the two eigenvalue bounds and then the square root gives the stated condition-number bound. □

At $N = M$, there are no missing rows and the deterministic bounds give $D_{\text{high}} = D_{\text{low}} = M/2$. Hence the theorem recovers $\kappa(\boldsymbol{R}_{\text{full}}) = 1$ exactly.

Theorem 8 quantifies the conditioning of ChebyLS under random straggling. In contrast to a generic coherence bound, it exploits both the responding workers and the complementary missing set, together with the deterministic structure of the complete Chebyshev frame. A numerical comparison with the general bound of Ipsen & Wentworth (2014, Theorem 4.1) is given in Table 1.

Finally, since $\mu \leq (2D-1)/M$, replacing μ by $(2D-1)/M$ weakens each concentration bound while preserving its validity. Substituting this estimate into the deterministic and probabilistic bounds above, and using the definitions of $\mathsf{U}_n(\delta)$ and $\mathsf{L}_n(\delta)$, yields Theorem 1.

N	Theorem 4	Theorem 1
800	2.610	2.188
1000	2.213	1.909
1200	2.001	1.729
1500	1.816	1.384
1800	1.701	1.175
1900	1.672	1.106
1950	1.659	1.070
1980	1.651	1.044
1990	1.648	1.033
2000	1.646	1.000

Table 1: Condition-number upper bounds for $M = 2000$, $D = 20$, and $\delta = 0.1$. Both bounds are evaluated at confidence at least $1 - 2\delta = 0.8$.

B Exact Recovery Probability of RepApprox

Proof of Theorem 2. Let E_i denote the failure event that task $i \in [K]$ is completely missing from the N responses. Because task i is assigned to exactly c servers, E_i occurs if and only if all N responses are drawn from the remaining $M - c$ servers in the cluster. More generally, the probability that a specific subset of j tasks is completely missing is the probability that all N responses are drawn from the $M - jc$ servers not computing those tasks,

$$\Pr\left(\bigcap_{k=1}^{j} E_{i_k}\right) = \frac{\binom{M-jc}{N}}{\binom{M}{N}}$$

where we use the standard convention that $\binom{a}{b} = 0$ if $a < b$. By the Inclusion-Exclusion Principle, the exact probability of successfully recovering all K tasks is the complement of the union of all failure events,

$$p_{\text{exact}} = 1 - \Pr\left(\bigcup_{i=1}^{K} E_i\right) = \sum_{j=0}^{K} (-1)^j \binom{K}{j} \frac{\binom{M-jc}{N}}{\binom{M}{N}}$$

While exact, this combinatorial summation is difficult to invert for system design. To find a more tractable lower bound, we apply the Union Bound over the single-task failure probabilities,

$$\Pr\left(\bigcup_{i=1}^{K} E_i\right) \leq \sum_{i=1}^{K} \Pr(E_i) = K \frac{\binom{M-c}{N}}{\binom{M}{N}}$$

By expanding the combinations,

$$\frac{\binom{M-c}{N}}{\binom{M}{N}} = \frac{(M-c)!N!(M-N)!}{N!(M-c-N)!M!} = \prod_{l=0}^{c-1} \frac{M-N-l}{M-l} \leq \left(1-\frac{N}{M}\right)^c$$

Substituting this into the complement of the Union Bound yields the final lower bound

$$p_{\text{exact}} \geq 1 - K\left(1-\frac{N}{M}\right)^c$$

□

C Numerical Experiments Details and Further Discussion

In our numerical experiments, we evaluate the coding frameworks by explicitly defining the arithmetic precision of the master and worker computations. The initial data encoding, the target model parameters, and the direct centralized evaluation (which serves as the reference ground truth $\boldsymbol{y}$) are computed using double-precision (`float64`). To observe the impact of finite-precision arithmetic during distributed inference, all simulated worker-side evaluations are cast to and computed in single-precision (`float32`). Due to this precision difference, the relative inference error for an exact algebraic recovery is fundamentally bounded below by the single-precision limit of $\approx 10^{-7}$.

For all experiments, we set $K = 10$ and $M = 100$. The target cross-network of L layers inference function is a polynomial of degree $L+1$, and the master utilizes an encoding polynomial of degree $K-1$, yielding a composite polynomial of degree $D-1 = (L+1)(K-1)$. In particular, if $L = 2$, the recovery threshold D of our experiment is 28 (see Figure 3). We sweep the number of responding servers N from the threshold limit ($N = D$) up to the full capacity ($N = M$). For each value of N, we perform 100 independent trials where the responding servers are selected uniformly at random. To account for the heavy-tailed variance, we report the median relative inference error alongside the 95% confidence intervals.

Cross-Only DCN Protocol: To isolate the numerical stability of polynomial decoding, we deploy the strictly polynomial Cross Network across the M servers using Chebyshev evaluation nodes, varying the network depth across $L \in \{2, 3, 4\}$ layers. We evaluate the relative recovery error of our overdetermined ChebyLS decoder by comparing it against both classical exact decoders (Threshold and BoX Chebyshev) and approximation-based schemes (BACC, BSACC, and our proposed RepApprox) (see Figure 3).

Parallel DCN v2 Protocol: To comprehensively evaluate our framework on a heterogeneous architecture, we deploy the full parallel DCN v2 model across depths of $L \in \{2, 3, 4\}$ layers. We test a suite of 12 decoding combinations, categorized into Hybrid and Uniform schemes.

- **Hybrid Schemes:** We explicitly distinguish the network's algebraic structures, applying exact polynomial decoding to the Cross branch and approximation-based decoding to the Deep branch. We evaluate all 9 pairings of the exact polynomial decoders (Threshold, BoX, and ChebyLS) with the deep branch decoders (BACC, BSACC, and RepApprox).
- **Uniform Schemes:** We treat the entire Parallel DCN v2 architecture as a general function. That is, we evaluate end-to-end distributed inference using BACC, BSACC, and RepApprox.

For combinations utilizing BACC or BSACC for the deep branch, the input data is coded and deployed across the entire $M = 100$ servers. For combinations incorporating the **RepApprox** framework, the cluster is partitioned into two dedicated pools: a coded pool ($M_{approx} = 50$) evaluating the encoded inputs, and a repetition pool ($M_{rep} = 50$) evaluating exact, uncoded tasks partitioned into $K = 10$ groups (with $c = 5$). For coded pool, we employ BSACC in our experiment but any approximation-based coded computing scheme with bounded error works.

During the decoding phase, the master node receives a total of N random responses. For the hybrid schemes, the Cross branch outputs are reconstructed via the respective Chebyshev decoder. For the deep branch, if RepApprox is utilized, the master executes the splicing logic: the exact `float32` evaluation is retained if at least one server from repetition group i responds, while the bounded-error approximation fallback is computed strictly for tasks where all $c = 5$ redundant servers fail.

We detail the DCN v2 model architecture for our experiment and our experiment design below.

Model Achitecture

We evaluate our coded computing frameworks on the parallel Deep & Cross Network v2 (DCN v2) (Wang et al., 2021) trained on the MovieLens 100K dataset (943 users, 1,682 items). Each user ID u and movie ID m are mapped to dense 16-dimensional embedding vectors $\boldsymbol{e}_u, \boldsymbol{e}_m \in \mathbb{R}^{16}$. Concatenating these embeddings yields the input feature vector $\boldsymbol{x}^{(0)} = [\boldsymbol{e}_u^\top, \boldsymbol{e}_m^\top]^\top \in \mathbb{R}^d$ with dimension $d = 32$. The network evaluates $\boldsymbol{x}^{(0)}$ across two parallel pathways:

- **Polynomial Cross Network (f_{cross}):** The cross branch consists of L stacked cross layers with dimension $d = 32$. Starting from $\boldsymbol{x}^{(0)}$, the representations are updated recursively for $\ell = 0, \ldots, L-1$ via

$$\boldsymbol{x}^{(\ell+1)} = \boldsymbol{x}^{(0)} \odot \Big(\boldsymbol{W}_\ell \boldsymbol{x}^{(\ell)} + \boldsymbol{b}_\ell \Big) + \boldsymbol{x}^{(\ell)},$$

  where $\boldsymbol{W}_\ell \in \mathbb{R}^{d \times d}$ and $\boldsymbol{b}_\ell \in \mathbb{R}^d$ are trainable weights and biases, and $\odot$ denotes element-wise multiplication. We set the pathway output as $f_{\text{cross}}(\boldsymbol{x}^{(0)}) \triangleq \boldsymbol{x}^{(L)} \in \mathbb{R}^d$. Expanding $\boldsymbol{x}^{(L)}$ via the previous layer yields

$$\begin{aligned}
\boldsymbol{x}^{(L)} &= \boldsymbol{x}^{(0)} \odot \Big(\boldsymbol{W}_{L-1} \boldsymbol{x}^{(L-1)} + \boldsymbol{b}_{L-1} \Big) + \boldsymbol{x}^{(L-1)} \\
&= \boldsymbol{x}^{(0)} \odot \Big(\boldsymbol{W}_{L-1} \big(\boldsymbol{x}^{(0)} \odot (\boldsymbol{W}_{L-2} \boldsymbol{x}^{(L-2)} + \boldsymbol{b}_{L-2}) + \boldsymbol{x}^{(L-2)} \big) + \boldsymbol{b}_{L-1} \Big) \\
&\quad + \big(\boldsymbol{x}^{(0)} \odot (\boldsymbol{W}_{L-2} \boldsymbol{x}^{(L-2)} + \boldsymbol{b}_{L-2}) + \boldsymbol{x}^{(L-2)} \big).
\end{aligned}$$

  Continuing this expansion down to the base input $\boldsymbol{x}^{(0)}$ shows that $\boldsymbol{x}^{(0)}$ appears multiplicatively L times in the leading interaction alongside the initial linear input $\boldsymbol{x}^{(0)}$, demonstrating that $f_{\text{cross}}(\boldsymbol{x}^{(0)}) \triangleq \boldsymbol{x}^{(L)}$ is an exact multivariate polynomial of degree $L + 1$.

- **Non-Polynomial Deep Network (f_{deep}):** The deep branch is an L-layer Multilayer Perceptron (MLP) with hidden dimensions $[d_1, \ldots, d_L]$ (with $d_0 = d_L = d = 32$). Starting from $\boldsymbol{h}^{(0)} = \boldsymbol{x}^{(0)}$, the representations propagate as

$$\boldsymbol{h}^{(\ell)} = \sigma \Big(\boldsymbol{W}_{d,\ell}\, \boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}_{d,\ell} \Big), \quad \ell = 1, \ldots, L,$$

  where $\boldsymbol{W}_{d,\ell} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$, $\boldsymbol{b}_{d,\ell} \in \mathbb{R}^{d_\ell}$, and $\sigma(z) = \max\{0, z\}$ denotes the element-wise ReLU activation. In composite form, the deep pathway output $f_{\text{deep}}(\boldsymbol{x}^{(0)}) \triangleq \boldsymbol{h}^{(L)} \in \mathbb{R}^d$ is

$$f_{\text{deep}}(\boldsymbol{x}^{(0)}) = \sigma\Big(\boldsymbol{W}_{d,L} \big(\cdots \sigma(\boldsymbol{W}_{d,1} \boldsymbol{x}^{(0)} + \boldsymbol{b}_{d,1}) \cdots \big) + \boldsymbol{b}_{d,L} \Big).$$

  The function f_{deep} is algebraically non-polynomial. In our experiments, we sweep $L \in \{2, 3, 4\}$, configuring hidden dimensions as $[16, 32]$ for $L = 2$, $[16, 16, 32]$ for $L = 3$, and $[16, 16, 16, 32]$ for $L = 4$.

The scalar model prediction combines both pathways via linear projection

$$y(\boldsymbol{x}^{(0)}) = f_{\text{cross}}(\boldsymbol{x}^{(0)})^\top \boldsymbol{v}_{\text{cross}} + f_{\text{deep}}(\boldsymbol{x}^{(0)})^\top \boldsymbol{v}_{\text{deep}} + b,$$

where $\boldsymbol{v}_{\text{cross}}, \boldsymbol{v}_{\text{deep}} \in \mathbb{R}^{32}$ and $b \in \mathbb{R}$. When interpolating K inputs $\{\boldsymbol{x}_k\}_{k=1}^K$ (where each $\boldsymbol{x}_k = \boldsymbol{x}_k^{(0)}$) with a degree-$(K-1)$ polynomial $\boldsymbol{u}(z)$, the composed cross target $h(z) = f_{\text{cross}}(\boldsymbol{u}(z))^\top \boldsymbol{v}_{\text{cross}}$ is a polynomial of degree $(L+1)(K-1) = D-1$. This yields a recovery threshold of $D = (L+1)(K-1)+1$ (e.g., $D = 28$ for $L = 2$, $D = 37$ for $L = 3$, and $D = 46$ for $L = 4$ when $K = 10$).

Experiment Design

Input: Batch of input vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K]$ where $\boldsymbol{x}_i \in \mathbb{R}^{32}$ with $K = 10$, total number of servers $M = 100$, number of responding servers N ($D \le N \le M$), trained DCN pathways f_{cross}, f_{deep}, final bias b, projection vectors $\boldsymbol{v}_{\text{cross}}$, $\boldsymbol{v}_{\text{deep}}$.

Ground Truth & Notation: Let $\boldsymbol{y}_{\text{true}} \in \mathbb{R}^K$ denote the inference outputs obtained by directly evaluating the trained model on $\boldsymbol{X}$ without any distributed computing scheme, computed strictly using double-precision (`float64`) arithmetic. Let $\mathcal{B} = \{\beta_k\}_{k=1}^K$ denote the set of K data evaluation nodes. Let $\mathcal{A} = \{\alpha_m\}_{m=1}^N$ denote the randomly sampled set of responding server evaluation nodes. The vector of finite-precision worker responses evaluated over $\mathcal{A}$ is denoted as $\hat{\boldsymbol{y}}_{\text{method},\mathcal{A}} \in \mathbb{R}^N$. The final reconstructed target computations evaluated over $\mathcal{B}$ are denoted with a hat as $\hat{\boldsymbol{y}}_{\text{method},\mathcal{B}} \in \mathbb{R}^K$.

Output: Relative inference error across varying response thresholds N.

E1. (Evaluation Nodes Initialization)

- For Threshold, BoX, ChebyLS:

$$\boldsymbol{\beta}_{\text{cheby}} = \left[\beta_{\text{cheby},i} = \cos\left(\frac{(2i-1)\pi}{2K}\right) \text{ for } i = 1, \ldots, K\right],$$
$$\boldsymbol{\gamma}_{\text{cheby}} = \left[\gamma_{\text{cheby},i} = \cos\left(\frac{(2i-1)\pi}{2M}\right) \text{ for } i = 1, \ldots, M\right].$$

- For BACC and BSACC, we use $\boldsymbol{\beta}_{\text{cheby}}$ for data evaluation points but with the following server evaluation points:

$$\boldsymbol{\gamma}_{\text{chebyv2}} = [\gamma_{\text{chebyv2},i} = \cos(i\pi/M) \text{ for } i = 1, \ldots, M].$$

- For ALCC:

$$\boldsymbol{\beta}_{\text{alcc}} = [\beta_{\text{alcc},k} = \exp(2i\pi \boldsymbol{j}/K) \text{ for } i = 1, \ldots, K],$$
$$\boldsymbol{\gamma}_{\text{alcc}} = [\gamma_{\text{alcc},i} = \exp(2i\pi \boldsymbol{j}/M) \text{ for } i = 1, \ldots, M].$$

(Note: $\boldsymbol{j}$ represents the imaginary unit).

- For RepApprox, we use $\boldsymbol{\beta}_{\text{cheby}}$ for data evaluation points. We fix the replication factor c, set $M_{\text{rep}} = cK$, and set the approximation pool $M_{\text{approx}} = M - cK$. Then, we set the server evaluation points as:

$$\boldsymbol{\gamma}_{\text{repapprox}} = [\gamma_{\text{repapprox},i} = \cos(i\pi/M_{\text{approx}}) \text{ for } i = 1, \ldots, M_{\text{approx}}].$$

E2. (Master Node Encoding) Pre-compute the encoded inputs for each server:

- Construct the degree $K-1$ polynomial $\boldsymbol{u}_{\text{cheby}}(z) \in \mathbb{R}^{32}[z]$ such that $\boldsymbol{u}_{\text{cheby}}(\beta_{\text{cheby},i}) = \boldsymbol{x}_i$ for $i = 1, \ldots, K$ using Lagrange interpolation in `float64`. The master node transmits $\boldsymbol{u}_{\text{cheby}}(\gamma_{\text{cheby},i})$ to server i in `float32`.

- Construct the degree $K-1$ polynomial $\boldsymbol{u}_{\text{alcc}}(z) \in \mathbb{C}^{32}[z]$ such that $\boldsymbol{u}_{\text{alcc}}(\beta_{\text{alcc},i}) = \boldsymbol{x}_i$ for $i = 1, \ldots, K$ using Lagrange interpolation in `complex128`. The master node transmits $\boldsymbol{u}_{\text{alcc}}(\gamma_{\text{alcc},i})$ to server i in `complex64`.
- Construct the interpolant $\boldsymbol{u}_{\text{approx}}(z) \in \mathbb{R}^{32}$ such that $\boldsymbol{u}_{\text{approx}}(\beta_{\text{cheby},i}) = \boldsymbol{x}_i$ for $i = 1, \ldots, K$ using Barycentric rational or natural cubic B-spline functions in `float64`. The master node transmits $\boldsymbol{u}_{\text{approx}}(\gamma_{\text{approx},i})$ to server i in `float32`, where $\gamma_{\text{approx},i} \in \{\gamma_{\text{chebyv2},i}, \gamma_{\text{repapprox},i}\}$ depending on the selected approximation scheme.

E3. (Server Computation) For each server $i \in \{1, \ldots, M\}$ in parallel, evaluate the pathways using single-precision arithmetic (`float32` or `complex64`) and compute the scalar responses:

- *Hybrid Evaluation Path:* Compute the isolated scalar projections:

$$\begin{aligned}\hat{y}^{(i)}_{\text{cross}} &= \left(f_{\text{cross}}\left(\boldsymbol{u}_{\text{cheby}}(\gamma_{\text{cheby},i})\right)\right)^{\top} \boldsymbol{v}_{\text{cross}},\\ \hat{y}^{(i)}_{\text{cross_alcc}} &= \left(f_{\text{cross}}\left(\boldsymbol{u}_{\text{alcc}}(\gamma_{\text{alcc},i})\right)\right)^{\top} \boldsymbol{v}_{\text{cross}},\\ \hat{y}^{(i)}_{\text{deep}} &= \left(f_{\text{deep}}\left(\boldsymbol{u}_{\text{approx}}(\gamma_{\text{approx},i})\right)\right)^{\top} \boldsymbol{v}_{\text{deep}}.\end{aligned}$$

  Server i returns the scalar tuple $(\hat{y}^{(i)}_{\text{cross}}, \hat{y}^{(i)}_{\text{deep}})$ or $(\hat{y}^{(i)}_{\text{cross_alcc}}, \hat{y}^{(i)}_{\text{deep}})$.
- *Pure Evaluation Path:* Compute the unified scalar response:

$$\hat{y}^{(i)}_{\text{comb}} = \left(f_{\text{cross}}\left(\boldsymbol{u}_{\text{approx}}(\gamma_{\text{approx},i})\right)\right)^{\top} \boldsymbol{v}_{\text{cross}} + \left(f_{\text{deep}}\left(\boldsymbol{u}_{\text{approx}}(\gamma_{\text{approx},i})\right)\right)^{\top} \boldsymbol{v}_{\text{deep}}.$$

  Server i returns the scalar $\hat{y}^{(i)}_{\text{comb}}$.

(Note: Either path has similar computational complexity as each server applies f_{cross} and f_{deep} to the received vector and multiplies with $\boldsymbol{v}_{cross}$ and $\boldsymbol{v}_{deep}$, respectively. In Pure Evaluation Path, only one extra scalar addition is needed.)

E4. (Master Node Decoding & Aggregation) For $N = D, \ldots, M$:

E4.1. Response Collection: Simulate a random straggler model by sampling an index set $\mathcal{I} \subseteq \{1, \ldots, M\}$ of size N uniformly at random. Let $\mathcal{A} = \{\alpha_i\}_{i=1}^{N} \subseteq \{\gamma_i\}_{i=1}^{M}$ denote the corresponding set of responding evaluation nodes. The collected finite-precision responses over these nodes are $\hat{\boldsymbol{y}}_{\text{cross},\mathcal{A}}$, $\hat{\boldsymbol{y}}_{\text{deep},\mathcal{A}}$, and $\hat{\boldsymbol{y}}_{\text{comb},\mathcal{A}}$.

E4.2. Hybrid Decoding Phase:

E4.2.1. Cross Branch:

- *For Exact Solvers (Threshold, BoX, ALCC):* Define a subset $\mathcal{R} \subset \mathcal{A}$ of size D. For Threshold and ALCC, $\mathcal{R}$ is selected uniformly at random. For Threshold, we consider the Chebyshev-Vandermonde decoding matrix $\boldsymbol{A}_{\mathcal{R}} = \boldsymbol{R}_{\mathcal{R}}$, whereas for ALCC, we consider the Vandermonde decoding matrix $\boldsymbol{A}_{\mathcal{R}} = \boldsymbol{V}_{\mathcal{R}}$. For BoX, we sample $X_{\text{samples}} = 200$ random size-D subsets of $\mathcal{A}$ and select the subset $\mathcal{R}$ with the smallest condition number Chebyshev-Vandermonde decoding matrix $\boldsymbol{A}_{\mathcal{R}} = \boldsymbol{R}_{\mathcal{R}}$. Recover the cross pathway polynomial coefficients $\hat{\boldsymbol{c}}$ by solving the square linear system:

$$\boldsymbol{A}_{\mathcal{R}}\hat{\boldsymbol{c}} = \hat{\boldsymbol{y}}_{\text{cross},\mathcal{R}}$$

- *For Least Squares (ChebyLS):* Retain the full set of N responses in $\mathcal{A}$. Recover the cross pathway polynomial coefficients $\hat{\boldsymbol{c}}$ by solving the overdetermined linear system with Chebyshev-Vandermonde decoding matrix $\boldsymbol{A}_{\mathcal{A}} = \boldsymbol{R}_{\mathcal{A}}$,

$$\boldsymbol{A}_{\mathcal{A}}\hat{\boldsymbol{c}} = \hat{\boldsymbol{y}}_{\text{cross},\mathcal{A}}$$

- Compute the target computations $\hat{\boldsymbol{y}}_{\text{cross},\mathcal{B}}$ evaluated over the data nodes $\mathcal{B}$.

E4.2.2. Deep Branch:

- *(BACC or BSACC):* Recover the deep pathway target computations $\hat{\boldsymbol{y}}_{\text{deep},\mathcal{B}} \in \mathbb{R}^K$ by interpolating the responses $\hat{\boldsymbol{y}}_{\text{deep},\mathcal{A}}$ via Berrut or B-spline functions.
- *(RepApprox):* Let G_k denote the repetition group (of size $c = 5$) assigned to task $k \in \{1, \ldots, K\}$. If at least one server from G_k has responded (i.e., $G_k \cap \mathcal{I} \neq \emptyset$), set the target computation $\hat{y}_{\text{deep},\mathcal{B},k}$ to the exact scalar response provided by that server. Otherwise, compute a fallback approximation for $\hat{y}_{\text{deep},\mathcal{B},k}$ by interpolating the responses from $\mathcal{I} \setminus \bigcup_k G_k$ via B-spline functions.

E4.2.3. Compute the final hybrid prediction: $\hat{\boldsymbol{y}}_{\text{hybrid}} = \hat{\boldsymbol{y}}_{\text{cross},\mathcal{B}} + \hat{\boldsymbol{y}}_{\text{deep},\mathcal{B}} + b \cdot \mathbf{1}$.

E4.3. Pure Decoding Phase:

- *(BACC or BSACC):* Recover the pure approximations $\hat{\boldsymbol{y}}_{\text{approx},\mathcal{B}} \in \mathbb{R}^K$ by interpolating the responses $\hat{\boldsymbol{y}}_{\text{comb},\mathcal{A}}$ via Berrut or B-spline functions.
- *(RepApprox):* Let G_k denote the repetition group (of size $c = 5$) assigned to task $k \in \{1, \ldots, K\}$. For each task k, if $G_k \cap \mathcal{I} \neq \emptyset$, set $\hat{y}_{\text{approx},\mathcal{B},k}$ to the exact combined response. Otherwise, compute the fallback approximation via B-spline interpolation from $\mathcal{I} \setminus \bigcup_k G_k$.
- Compute the final pure prediction $\hat{\boldsymbol{y}}_{\text{pure}} = \hat{\boldsymbol{y}}_{\text{approx},\mathcal{B}} + b \cdot \mathbf{1}$.

E5. (Relative Error Computation): Compute the relative inference error for each respective method

$$\varepsilon = \frac{||\hat{\boldsymbol{y}}_{\text{method}} - \boldsymbol{y}_{\text{true}}||_2}{||\boldsymbol{y}_{\text{true}}||_2}$$

Remark 9 (Computational Complexity: ChebyLS vs. Best-of-X (BoX))**.** *We compare the computational complexity of ChebyLS against BoX. Solving the overdetermined linear system in ChebyLS via standard Householder QR factorization requires $\mathcal{O}(ND^2)$ operations. In contrast, evaluating the condition number across X candidate size-D subsets in BoX requires $\mathcal{O}(XD^3)$ operations, followed by $\mathcal{O}(D^3)$ operations to invert the chosen submatrix. Comparing the dominant terms, BoX matches the computational cost of ChebyLS only when $X \approx N/D$. In our experiments (see Figure 5 when $N = 50$ and $D = 37$), we granted BoX a massive computational advantage by setting $X = 200$, incurring roughly $X(D/N) \approx 150$ times the computational cost of ChebyLS. Despite this advantage, Figure 5 demonstrates that the condition number of BoX plateaus around $\approx 10^3$, failing to match the condition number achieved by ChebyLS.*

Figure 3 illustrates the relative inference error for the Cross-only Network across varying depths ($L \in \{2, 3, 4\}$). At the classical exact recovery threshold ($N = D$), all polynomial decoders exhibit a high relative error due to the ill-conditioned square Chebyshev-Vandermonde matrix. However, as the number of available worker responses increases ($N > D$), the fundamental limitations of square-system decoders become apparent. Threshold Chebyshev discards the algebraically redundant responses, causing its error to remain perpetually high. While BoX Chebyshev attempts to improve recovery by searching for better-conditioned square subsystems, this approach scales poorly in practice; evaluating even a fraction of the $\binom{N}{D}$ combinatorial candidate subsets incurs computational overhead, yet still fails to match the stability of an overdetermined system (see Figure 5). In contrast, our proposed **ChebyLS** decoder integrates all N available responses into a single least-squares evaluation. By suppressing the finite-precision noise amplification, ChebyLS steadily drives the relative inference error downward as N increases, reliably approaching the single-precision machine limit ($\approx 10^{-7}$) across all tested network depths.

Figure 4 evaluates the end-to-end inference error for the Parallel DCN v2 architecture. Focusing first on the existing approximation-based methods (BACC and BSACC) without our RepApprox framework, we observe that the overall system performance is heavily bottlenecked by the subnetwork with the higher relative error, the non-polynomial Deep Network. Because the bounded approximation error of the Deep branch dominates the final composite output, pairing a highly stable exact polynomial decoder like ChebyLS with BACC or BSACC fails to yield a significant end-to-end improvement. As a result, these alternative

hybrid combinations trace an error floor that is practically indistinguishable from the purely approximate, uniform baselines.

This performance bottleneck directly motivates our **RepApprox** framework. While RepApprox is included in Figure 3 and 4, we defer its discussion to Appendix D, where we empirically analyze its exact recovery probability and quantify the number of tasks that actually require the approximation fallback. Additionally, we provide the end-to-end inference error for the Parallel DCN v2 architecture at depths of 2 and 4 layers in Figure 6.

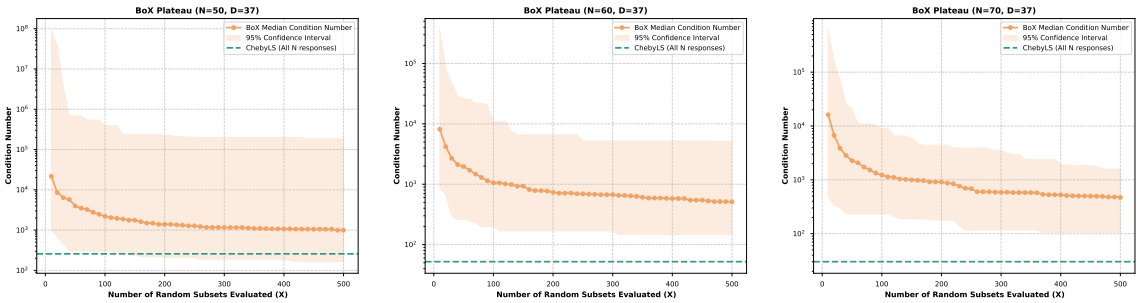

Figure 5: Empirical condition number of the Best-of-X (BoX) Chebyshev decoder as a function of evaluated candidate subsets (X) for a 3-Cross network ($D = 37$). Across varying response pool sizes ($N \in \{50, 60, 70\}$), the condition number of the optimally selected square subsystem plateaus.

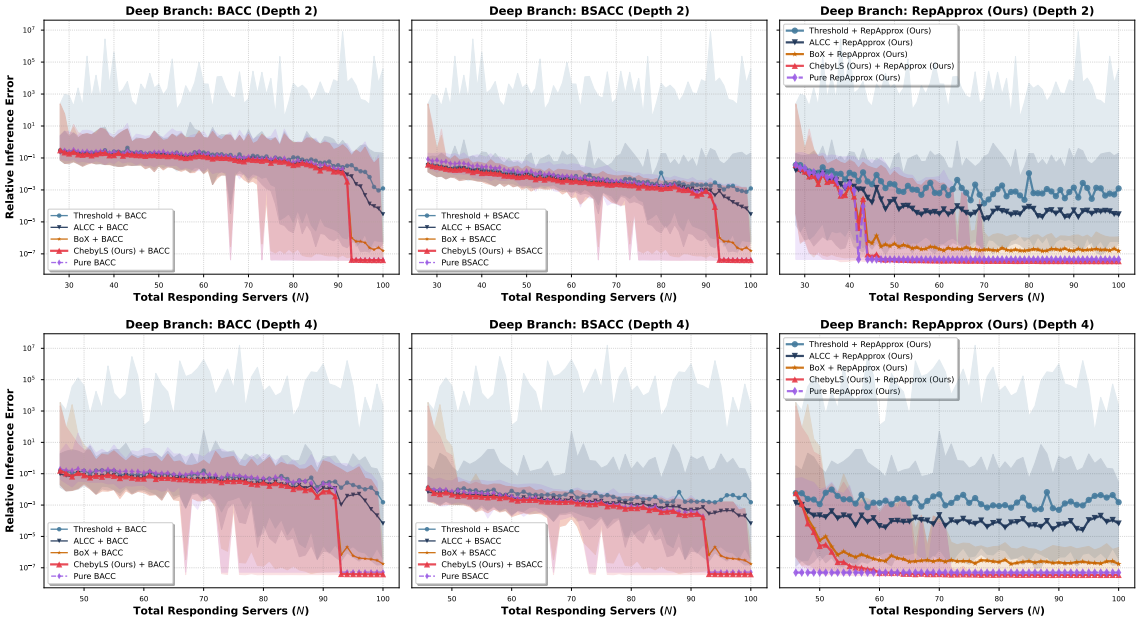

Figure 6: End-to-end relative inference error for the Parallel DCN v2 with depth 2 and 4 ($K = 10$, $M = 100$).

D FURTHER EXPERIMENTS ON REPAPPROX

To empirically validate the exact recovery guarantees of Theorem 2 and quantify the approximation burden of our hybrid architecture, we simulate the RepApprox splicing logic under a uniform random straggler model. We evaluate a distributed system of $M = 100$ servers computing $K = 10$ tasks. To analyze the trade-off between replication redundancy and approximation fidelity, we sweep the replication factor across $c \in \{4, 5, 6, 7\}$, which allocates $M_{rep} = cK$ servers to the exact repetition pool. For each configuration, we execute $100,000$ independent trials, drawing N responding servers uniformly without replacement for $N \in \{30, 40, \ldots, 100\}$. In each trial, we index the intersection of the N responses with the assigned task groups to determine if any of the K tasks are completely missing from the repetition pool.

The results of this ablation demonstrate two core systemic advantages. First, the empirical probability matches the combinatorial formula derived in Theorem 2 across all values of N and c. Second, and more importantly, the simulation proves that RepApprox exhibits exceptional graceful degradation. By tracking the median and 95% Empirical Quantile Range of the number of fallback tasks triggered per trial, we observe that systemic failures are highly contained. For example, configuring the system with $c = 5$ and waiting for only half of the cluster to respond ($N = 50$) yields a median of 0.0 approximated tasks, with the 95% Empirical Quantile Range bounded between $[0.0, 1.0]$.

As observed in the empirical evaluations in Figures 3 and 4, the median relative inference error of the **RepApprox** schemes exhibits a distinct phase transition as the number of responding servers N increases. When the response pool is small (e.g., $N = 30$), the RepApprox scheme yields a relative error that is comparable to the purely approximation-based methods. However, as N increases beyond 60, the median relative error drops precipitously, seamlessly achieving the single-precision machine limit ($\approx 10^{-7}$) associated with exact algebraic recovery.

This empirical behavior is perfectly consistent with the exact recovery probabilities formalized in Theorem 2 and simulated in Table 3 for the $c = 5$ configuration. At $N = 30$, the probability of exact recovery p_{exact} is merely 13.91%, forcing the system to rely heavily on the approximation fallback (with a 95% Empirical Quantile Range of $[0.0, 4.0]$ approximated tasks). Consequently, the median error at this threshold is naturally dominated by the approximation baseline. Conversely, when $N = 60$, p_{exact} jumps to 91.48%, and the 95% Empirical Quantile Range for the number of fallback tasks drastically tightens to $[0.0, 1.0]$. Because the system successfully recovers all tasks exactly in over 90% of the trials at this threshold, the median relative error is driven strictly by the exact evaluations, allowing it to hit the 10^{-7} floor. This directly validates the core design principle of RepApprox: it provides a robust, bounded-error safety net under severe straggling regimes ($N \leq 40$), while dynamically and reliably returning exact inference once a moderate fraction of the cluster ($N \geq 60$) responds.

Table 2: RepApprox reliability under random straggling for configuration $c = 4$ ($M_{rep} = 40$).

Responses	**Exact Recovery Probability**		**Approximated Tasks (Fallback)**	
(N)	Theoretical	Empirical	Median	95% Empirical Quantile Range
30	0.0487	0.0499	2.0	[0.0, 5.0]
40	0.2368	0.2356	1.0	[0.0, 3.0]
50	0.5294	0.5303	0.0	[0.0, 2.0]
60	0.7849	0.7835	0.0	[0.0, 1.0]
70	0.9315	0.9319	0.0	[0.0, 1.0]
80	0.9877	0.9879	0.0	[0.0, 0.0]
90	0.9995	0.9995	0.0	[0.0, 0.0]
100	1.0000	1.0000	0.0	[0.0, 0.0]

Table 3: RepApprox reliability under random straggling for configuration $c = 5$ ($M_{rep} = 50$).

Responses	**Exact Recovery Probability**		**Approximated Tasks (Fallback)**	
(N)	Theoretical	Empirical	Median	95% Empirical Quantile Range
30	0.1391	0.1402	2.0	[0.0, 4.0]
40	0.4470	0.4465	1.0	[0.0, 2.0]
50	0.7442	0.7445	0.0	[0.0, 1.0]
60	0.9148	0.9145	0.0	[0.0, 1.0]
70	0.9811	0.9810	0.0	[0.0, 0.0]
80	0.9979	0.9981	0.0	[0.0, 0.0]
90	1.0000	1.0000	0.0	[0.0, 0.0]
100	1.0000	1.0000	0.0	[0.0, 0.0]

Table 4: RepApprox reliability under random straggling for configuration $c = 6$ ($M_{rep} = 60$).

Responses	**Exact Recovery Probability**		**Approximated Tasks (Fallback)**	
(N)	Theoretical	Empirical	Median	95% Empirical Quantile Range
30	0.2739	0.2739	1.0	[0.0, 3.0]
40	0.6364	0.6361	0.0	[0.0, 2.0]
50	0.8718	0.8718	0.0	[0.0, 1.0]
60	0.9680	0.9683	0.0	[0.0, 1.0]
70	0.9950	0.9954	0.0	[0.0, 0.0]
80	0.9997	0.9998	0.0	[0.0, 0.0]
90	1.0000	1.0000	0.0	[0.0, 0.0]
100	1.0000	1.0000	0.0	[0.0, 0.0]

Table 5: RepApprox reliability under random straggling for configuration $c = 7$ ($M_{rep} = 70$).

Responses	**Exact Recovery Probability**		**Approximated Tasks (Fallback)**	
(N)	Theoretical	Empirical	Median	95% CI
30	0.4264	0.4280	1.0	[0.0, 2.0]
40	0.7759	0.7774	0.0	[0.0, 1.0]
50	0.9386	0.9381	0.0	[0.0, 1.0]
60	0.9884	0.9886	0.0	[0.0, 0.0]
70	0.9987	0.9988	0.0	[0.0, 0.0]
80	1.0000	1.0000	0.0	[0.0, 0.0]
90	1.0000	1.0000	0.0	[0.0, 0.0]
100	1.0000	1.0000	0.0	[0.0, 0.0]

E Inference Performance under Stochastic Server Response Times

While the preceding experiments evaluate inference accuracy as a function of a fixed number of responding workers N, in practical distributed serving pipelines worker response times are inherently random and asynchronous. To investigate this effect, we perform an experiment by modeling the response time T_i of worker

$i \in \{1, \ldots, M\}$ as a random variable comprising two components: a deterministic computation time t_{comp} required to evaluate the neural network forward pass, and a stochastic delay $\tau_i \sim \text{Exponential}(\mu)$ capturing system jitter, where μ^{-1} denotes the mean stochastic delay. Specifically,

$$T_i = t_{\text{comp}} + \tau_i, \quad \text{for } t \geq t_{\text{comp}}. \tag{1}$$

To keep the evaluation hardware-agnostic, we set $\mu^{-1} = 0.5\, t_{\text{comp}}$. By doing so, the service time can be written as $T_i = t_{\text{comp}}(1 + 0.5 Z_i)$, where $Z_i \sim \text{Exponential}(1)$ is a standard unit-rate exponential random variable. For this experiment specifically, we set $t_{\text{comp}} = 1$.

In this simulation, the master broadcasts the encoded queries to all $M = 100$ workers at $t = 0$. As wall-clock time elapses, workers return their evaluations asynchronously, and the master attempts reconstruction as soon as at least $D = 37$ responses have arrived. Figure 7 presents the resulting relationship between elapsed wall-clock latency and relative inference error across all distributed computing combinations.

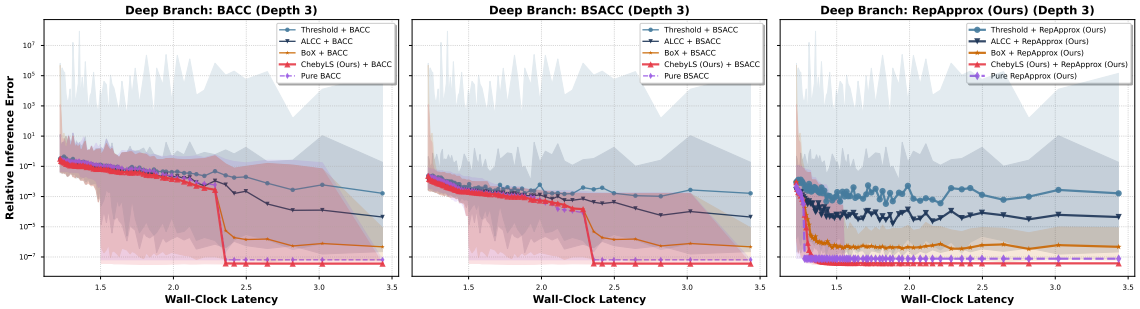

Figure 7: Relative inference error versus wall-clock latency under stochastic worker response times $T_i = t_{\text{comp}} + \tau_i$, $\tau_i \sim \text{Exponential}(\mu)$ with $t_{\text{comp}} = 1$ and $\mu^{-1} = 0.5 t_{\text{comp}} = 0.5$.

F AI Assistance Disclosure

In this work, we used generative AI tools for polishing writing, finding related works, research ideation and execution, drafting sections of the paper, and proving mathematical claims. We have not used generative AI tools for generating synthetic datasets. We have reviewed all AI-assisted work. Specifically, all AI-assisted mathematical proofs were rigorously verified for theoretical correctness, AI-polished text were manually reviewed and edited to ensure scientific accuracy, and AI-suggested related works were independently validated against primary literature. We take responsibility for the final content of this work.