

A APPENDIX

A.1 CODED COMPUTING

Inspired by the success of coding theory in communication over unreliable channels, *coded computing* has emerged as an efficient framework for distributed computation. It addresses key challenges in distributed systems, particularly the presence of stragglers or adversarial servers (Yu et al., 2017). Early work introduced coded computing for fundamental tasks such as polynomial evaluation (Yu et al., 2019; Fahim & Cadambe, 2021) and matrix multiplication (Yu et al., 2017; Jahani-Nezhad & Maddah-Ali, 2021). More recently, Moradi et al. (2024, 2025); Moradi & Maddah-Ali (2025) proposed a framework grounded in learning theory that extends coded computing to a wide range of functions, including deep neural networks, with provable resilience against stragglers and adversaries.

The central idea of coded computing is to assign each server a (linear) combination of the data, referred to as coded data, instead of the raw inputs. The number of coded symbols exceeds that of the original data, effectively over-representing the data. This redundancy is then leveraged to mitigate the effects of stragglers and adversarial behavior. Formally, suppose a master node aims to approximately compute a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ on a dataset $\{x_1, \dots, x_K\}$ using a cluster of N servers, some of which may be stragglers. The coded computing framework proceeds in three steps:

- (1) **Encoding:** The master node fits an encoder function $u_{\text{enc}} : \mathbb{R} \rightarrow \mathcal{X}$ to the set of points $\{(\alpha_i, x_i)\}_{i=1}^K$, where $\alpha_1 < \alpha_2 < \dots < \alpha_K \in \Omega \subset \mathbb{R}$ are referred to as *encoding points*. Thus,

$$\forall i \in [K], \quad u_{\text{enc}}(\alpha_i) \approx x_i. \quad (11)$$

The master node then generates N coded data by evaluating the encoder at another fixed set $\{\beta_j\}_{j=1}^N$ with $\beta_1 < \beta_2 < \dots < \beta_N \in \Omega \subset \mathbb{R}$, called *decoding points*:

$$\tilde{x}_j = u_{\text{enc}}(\beta_j), \quad j \in [N]. \quad (12)$$

Each coded point \tilde{x}_j is a combination of the original input dataset $\{x_i\}_{i=1}^K$, and is then assigned to server j .

- (2) **Computation:** Each server j computes $f(\tilde{x}_j)$ and returns the result to the master. Due to the presence of stragglers, some results may be missing. Let \mathcal{F} denote the set of indices corresponding to successfully returned results.
- (3) **Decoding:** Given the received outputs $\{f(\tilde{x}_v)\}_{v \in \mathcal{F}}$ from the non-straggler servers, the master node fits a decoder function $u_{\text{dec}} : \mathbb{R} \rightarrow \mathcal{Y}$ at the points $\{(\beta_v, f(\tilde{x}_v))\}_{v \in \mathcal{F}}$. Consequently,

$$\forall j \in [N], \quad u_{\text{dec}}(\beta_j) \approx f(u_{\text{enc}}(\beta_j)). \quad (13)$$

If the decoder $u_{\text{dec}}(\cdot)$ generalizes well, it can approximate $f(\cdot)$ on the original dataset $\{x_i\}_{i=1}^K$:

$$\hat{f}(x_i) \triangleq u_{\text{dec}}(\alpha_i) \approx f(u_{\text{enc}}(\alpha_i)) \approx f(x_i), \quad (14)$$

where the first approximation relies on the generalization ability of u_{dec} , and the second follows from (11).

The goal is to obtain an accurate estimate of the function $f(\cdot)$ on the input dataset. The key design choice in the coded computing scheme is selecting encoder and decoder functions that yield low estimation error.

Moradi et al. (2024) propose using smoothing splines (Wahba, 1975; 1990) as both encoder and decoder functions, fitted on $\{x_i\}_{i=1}^K$ and $\{f(\tilde{x}_v)\}_{v \in \mathcal{F}}$ with smoothing parameters λ_e and λ_d , respectively. More specifically, the decoder is obtained by solving the following optimization problem:

$$u_{\text{dec}}^* = \arg \min_{u \in \mathcal{H}^2(\Omega)} \frac{1}{|\mathcal{F}|} \sum_{v \in \mathcal{F}} \|u(\beta_v) - f(u_{\text{enc}}(\beta_v))\|^2 + \lambda_d \cdot \|u''\|_{L^2(\Omega)}^2, \quad (15)$$

where $\mathcal{H}^2(\Omega)$ denotes the second-order Sobolev space over Ω (i.e., functions with square-integrable derivatives up to order two), and $\|\cdot\|_{L^2(\Omega)}$ denotes the L^2 -norm on Ω .

Using the decoder function in (15), together with a careful choice of encoding points, decoding points, and smoothing parameter, Moradi et al. (2024) show that the mean squared estimation error of the coded computing scheme can be upper bounded as follows:

Theorem 1 (Moradi et al. (2024)). *Consider the coded computing framework with N servers and at most S stragglers. Suppose $\Omega = (-1, 1)$ and $f(\cdot)$ is μ -Lipschitz continuous. Then,*

$$\frac{1}{K} \sum_{i=1}^K \left\| \hat{f}(x_i) - f(x_i) \right\|^2 \leq C \left(\frac{S+1}{N} \right)^3 \| (f \circ u_{\text{enc}})'' \|_{L^2(\Omega)}^2 + \frac{2\mu^2}{K} \sum_{k=1}^K \| u_{\text{enc}}(\alpha_k) - x_k \|^2, \quad (16)$$

where $C > 0$ is a constant.

B PROOF OF LEMMA 1

Since in the coded-smoothing module $u_{\text{enc}}(\alpha_i) = x_i$, the second term in Theorem 1 vanishes. Setting $S = 0$ and applying the chain rule to $(f \circ u_{\text{enc}})''$ completes the proof of the lemma.

C COMPUTATIONAL COMPLEXITY OF CODED-SMOOTHING MODULE

Both the encoder and decoder functions are non-parametric. Therefore, the coded-smoothing module does not introduce any additional learnable parameters to the model. Moreover, the encoding and decoding steps are computationally efficient. This efficiency arises because fitting and evaluating smoothing splines can be performed in linear time using the B-spline basis representation (De Boor, 2001; Eilers & Marx, 1996). Specifically, if the input and output dimensions are d and m , respectively, the computational complexities of encoding and decoding steps (including both fitting and evaluating) are $\mathcal{O}((N+K) \cdot d)$ and $\mathcal{O}((N+K) \cdot m)$.

D EXPERIMENTAL DETAILS

D.1 SUPERVISED

For all supervised learning experiments, we trained models on CIFAR-10, CIFAR-100, and Tiny ImageNet datasets under consistent optimization settings. In the CIFAR experiments, the number of training epochs was set to 350, with an initial learning rate of 0.1. The learning rate was decayed by a factor of 10 at epochs 100 and 250 for both CIFAR-10 and CIFAR-100. For Tiny ImageNet, the learning rate decay was applied at epochs 100, 200, and 300. Stochastic Gradient Descent (SGD) with a momentum of 0.9 was used as the optimizer for all experiments. The batch size was set to 128.

D.1.1 HYPERPARAMETER SELECTION

For mixup, the mixing coefficient λ is sampled from a Beta distribution $\text{Beta}(\alpha, \alpha)$, where α is chosen according to the best-performing settings reported in prior works (Zhang et al., 2017; Verma et al., 2019). Specifically, we use $\alpha = 1.0$ for CIFAR-10 and CIFAR-100, and $\alpha = 0.2$ for TinyImageNet.

Training with the CODED-SMOOTHING method introduces two hyperparameters: μ and N . The parameter μ in (8) balances the contribution of the coded path in the overall loss, while N controls the accuracy of the estimation. From Lemma 1, the discrepancy between the original and coded path outputs decreases either as N increases or as $f(\cdot)$ becomes smoother. However, setting N too large makes the coded and original paths nearly identical, effectively collapsing the method to ERM. Empirically, we find that initializing with $N = K$ (the batch size, see Figure 2) and $\mu = 0.5$ yields the best trade-off between regularization strength and predictive accuracy. A detailed sensitivity analysis of N and μ is presented in Tables 5 and 6 in Appendix G.

Scheduling N . Fixing N during training causes saturation, as the gap between f and \hat{f} stops shrinking once a certain smoothness is reached. To address this, we gradually increase N from the batch size K to γK with $\gamma > 1$. We find $\gamma = 1.5$ works best, yielding two benefits: improved

coded-path accuracy (making it reliable for inference) and escaping training plateaus for further gains (see Figure 5a).

D.2 UNSUPERVISED

For the unsupervised experiments with WGAN, we used the following configuration: the training was performed for 100,000 iterations with a batch size of 64. The number of coded points (N) was set to 96. Moreover $\mu = 0.5$. The initial learning rate was 2×10^{-4} , and the critic was updated 5 times per generator step. We employed a gradient penalty coefficient $\lambda_{gp} = 10$, and the optimizer was Adam with betas (0.0, 0.9). For Inception Score (IS) computation, 50,000 samples were used. For FID computation, we followed the standard protocol by comparing the statistics of 50,000 generated images with the real dataset.

E COVARIATE SHIFT ROBUSTNESS

Table 4: Comparisons of accuracies (%) on out-of-distribution test data.

	CIFAR-10.1	CIFAR-10.2	CIFAR-10.C
ERM	86.5 ± 0.4	82.8 ± 0.1	72.7 ± 0.3
Mixup	88.9 ± 0.4	85.7 ± 0.2	78.2 ± 0.3
CODED-SMOOTHING (ours)	89.6 ± 0.5	86.4 ± 0.1	77.6 ± 0.2

F PERFORMANCE DURING TRAINING

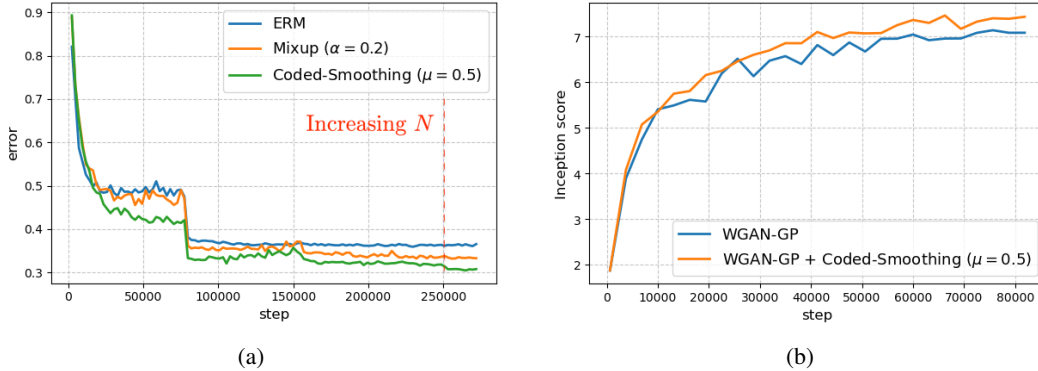


Figure 5: (a) TinyImageNet validation loss for different methods during training. (b) Comparison of Inception Score (IS) during training on the CIFAR-10 dataset. In WGAN-GP + CODED-SMOOTHING, the coded-smoothing module is applied to the generator of the GAN architecture.

G ABLATION STUDY

G.1 EFFECT OF NUMBER OF CODED SAMPLES (N)

Table 5: Test accuracy (%) of training with coded-smoothing module for different number of coded samples (N) on CIFAR-10 with batch size 128.

N	Acc
110	95.6
130	95.0
150	94.8
170	94.5
190	94.2

G.2 EFFECT OF μ

Table 6: Test accuracy (%) of training with coded-smoothing module for different μ on CIFAR-10.

μ	Acc
0.1	95.1
0.2	95.4
0.4	95.7
0.5	95.9
0.6	95.8
0.8	95.4
1.0	95.0

G.3 EFFECT OF BATCH SIZE AND NUMBER OF CODED SAMPLES IN RCI

Table 7: Comparison of accuracies of inference with coded path for the mode methods under Adversarial attack on CIFAR-10 dataset.

Batch Size	# Coded Samples (N)	No Attack	FGSM ($\epsilon = \frac{8}{255}$)	PGD (10 steps)
8	12	94.9	78.1	69.8
16	24	94.8	79.0	70.3
32	48	94.9	79.3	70.6
64	96	94.9	80.1	71.5
128	190	94.9	80.5	72.0