We thank the reviewers for all remarks/suggestions. We appreciate the acknowledgments on: problem formulation being timely (**R1**) and significant/valuable (**R3**); extensive analysis and experimental results provided (**R1**, **R2**, **R3**); novelty (**R1**) and improved redundancy over existing DRACO (**R3**); and good utilization of hierarchical voting (**R2**). As for the concerns/questions raised, we believe we successfully addressed every single one, as explained below.

Computational load (R1, R2) True, any coding requires redundant computation in return for better Byzantine protection. Our method offers great options to protect system with reasonable extra computation. As an example, our Bernoulli code with mean redundancy of 2 (two partitions per worker), a reasonable redundancy factor, achieves an 88% accuracy at unit time of 609 while uncoded SignSGD-MV reaches a 79% accuracy at a slower time of 750 and fails altogether to reach the 88% level with 3 out of 9 workers compromised (Table 1 in Supplementary Materials (SM)). Further, using reduced minibatch size, effective redundancy reduces to as low as 1.25, for example, while providing a solid 85% accuracy for a severe 40% Byzantine scenario (2 out of 5 compromised) as in Fig. A.3a of SM. Under the same condition, uncoded SignSGD-MV fails to give over a 40% accuracy, regardless of train time. Reducing minibatch size for uncoded scheme results in unacceptable performance (note that for our method, each worker extracts gradient from multiple data partitions so that the detrimental effect of reducing minibatch size is not felt in a significant way).

Additional experiments (R1, R2) We obtained new experimental results on full gradient + median (FGM) (Fig. 1a below) and also for no-attack scenarios for the severe 40% Byzantine case (Fig. 1abc). Our scheme with redundancy as low as $r_{\text{eff}} = 1.5$ outperforms FGM. While FGM requires no computational redundancy, it needs 32x more communication bandwidth than ours while performing worse. For the 20% Byzantine scenario ($b = 3$), FGM performs relatively better, but still falls well below ours. For the severe 40% Byzantine case, it is harder to achieve near-perfect protection but our schemes clearly perform better. We also ran experiments for a larger network, ResNet-50, as seen in Fig. 1b. Our scheme with redundancy $r_{\text{eff}} = 1.5$ is still effective here. Thanks to the suggestion by **R2**, we added independent Gaussian noise to all gradients corresponding to individual partitions before the signs are taken (in addition to Byzantine attacks on local majority signs). The proposed Bernoulli code can tolerate noisy gradient computation well, while uncoded signSGD-MV cannot, as shown in Fig. 1c for added noise with variance $\sigma^2 = 1e^{-8}$.

Optimal? (R1) This is a tough but highly relevant question. Rephrased, we ask: what is minimal code redundancy while providing (asymptotically) perfect Byzantine tolerance? From L273 of SM, minimum connection probability (proportional to redundancy) of random codes for full Byzantine protection must satisfy $p = f(k)/\sqrt{k}$ with $f(k)$ being a growing function of k, # of data partitions. Our choice $p \sim \sqrt{\log(k)/k}$ meets this, and thus is asymptotically optimal.

***What if # of data partitions k is not equal to # of workers n* (R1)** An interesting question. When $n \neq k$, the required connection probability for Byzantine tolerance in Lemma 1 behaves as $p \sim \sqrt{\log(k)/k}$; computational load at each worker is proportional to p and thus load can be lowered by increasing k. However, the local majority vote becomes less accurate with increasing k since each worker uses fewer data points (parameter S deteriorates; see S in equation 2 and Proposition/Definition 1). We feel a more effective solution is reducing mini-batch size as shown in A.4 in SM.

Versus Boolean computation (R1) Boolean computation on noisy gates optimizes # of gates for accuracy; we concern how to connect inputs to majority gates to tolerate Byzantines with minimal connections, an entirely different issue.

Clarifications (R1, R2) Yes, the master obtains the correct result for $\|\boldsymbol{m}\|_0 < \lfloor n/2 \rfloor$, according to proof of Lemma 2 (**R1**). Our intention in Lemma1 is to ensure c_i, the local majority vote, closely reflects sign of global gradient g (**R2**).

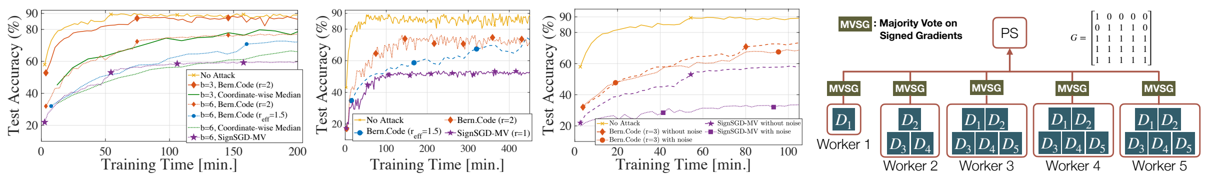
Minor correction (R1) Our scheme focuses only on quantized gradients; we will correct this and typos in equation (2).

Nonuniform computational load (R2) True, our worker loads are non-uniform, but load distribution for Bernoulli codes tends to uniform as n grows, due to concentration property of binomial distribution. The probability that individual load deviates from mean np by $n\varepsilon$ is less than $2e^{-2\varepsilon^2 n}$, as seen in Lemma C.5 of SM.

Solution not so novel (R3) We disagree. If **R3** implies majority voting is the gist of our method, we stress it is not. In any case, Election Coding is about redundant allocation of data partitions to workers in conjunction with hierarchical voting, a new framework that does not exist in coding/information theory community or machine learning community.

Attacks between voters and polling station (R3) No attacks can take place between voters and polling stations. Think of the voters (and their clones) actually going to the polling stations and casting ballots themselves with no possibility for voting fraud up to that point. In the real setup, voters are like data partitions while polling stations represent workers. The individual gradients are actually computed at the worker nodes with no exposed connections. See Fig. 1d below.

High level explanation and better example (R3, R1) Figs. 1&2 along with associated discussions provide high level understanding on suggested coding methodology for redundant distribution of data partitions to workers (also see Fig. 1d below for a new example); perhaps what **R3** is asking is a high-level description of distributed learning itself. It will be a simple thing to add this in the revision, given a chance. Thanks to **R1**'s request, Fig. 1d below gives an example code that perfectly tolerates $b = 1$ Byzantine on any node. We will switch to this example in the revision.



(a) Compare with median (b) ResNet-50 (c) Noisy computation (d) Perfect Byzantine-tolerant example

Figure 1: (a)-(c): Experimental results on CIFAR-10 dataset, for $n = 15$ and $b = 6$ (unless stated otherwise), averaged out over three trials; (d): Example of the suggested deterministic code that perfectly tolerates $b = 1$ Byzantine out of $n = 5$ nodes