Reviewer 1. 1 .*"more background review"* - Thanks for the references. We will add discussion of these papers. **2.*"return communication grows with W.."*** - In k-sparsification methods, the weight update is the sum of W k-sparsified gradients and can have Wk non-zero entries. In practice, the weight update is dense for large W, as shown in Figure 4. There is no way to losslessly compress this back-communication to under $O(Wk)$ bytes, and we are not aware of any lossy compression that achieves good performance in practice. **3. *"In practice how should the size of the Count Sketch table be set?"*** - Theory suggests the sketch size should be $O(k \log d)$, with $c = O(k)$ columns and $r = O(\log_c d)$ rows. The exact values must be found experimentally. ***"I expect if the model parameters themselves are sparse then the gradients are more likely to be sparse"*** - In general, sparsity of model parameters doesn't imply sparsity of gradients. For example, consider squared loss $(y - \mathrm{w}^\top \mathrm{x})^2$ with a sparse w, the gradient is $-2(y - \mathrm{w}^\top \mathrm{x})\mathrm{x}$, which might not be sparse. We make no assumptions about the sparsity of the gradients, and for the models and loss functions we use (and that are commonly used), the gradients are dense.

Reviewer 2.(A) ***"does it reduce training time.."*** - We have not yet benchmarked overall running time on realistic network topologies, though we are actively working in this direction. Sketching does impose some computational overhead, and with our current implementation we expect a speedup in overall run time when communication cost is about as high (or higher) than the computation cost. This is realistic for a large network (200M parameters) even over a fast (10Gbps) network. In the regime of many workers (e.g. federated learning), where our algorithm performs best theoretically, network can be as slow as a few Mbps. **(B)** ***using asynchronous updates*** - We have not experimented with asynchronous updates, but we expect that the benefits asynchrony brings to regular SGD would also apply to Sketched SGD. We believe our theoretical results could be extended to the asynchronous case, but we leave this to future work.

Reviewer 3. 1. *"Bounded second moment and variance assumption"* - A bounded second moment implies bounded variance (via variational definition of variance) , so $\sigma \leq B$. However note that when you average W stochastic gradients, the variance reduces by factor W, but the second moment bound is still the same. We use different parameters for variance & second moment bounds so that it is easy to argue how the quantities in the upper bound change when we average stochastic gradients. See remark 3 in the paper.
2.*"$k \log(dT/\delta) > d$ for sufficiently large T..."* - $T > \Omega(e^d)$ is not realistic for even our smaller model ($d \approx 10^7$).
3. *"Can't one send the compressed gradients back..?"* The $O(W)$ per-worker communication cost is not a technicality: if each worker sends k non-zero gradient elements, the weight update (which is the sum of these sparse gradients) will have up to kW non-zero elements. In practice, the weight update quickly does become dense, as shown in Figure 4. In principle, one could lossily compress the back communication, but we are not aware of any methods that do so while maintaining high performance.
4. *".. you only send back the k elements that were modified right?"* - Correct, $\tilde{g}_t$ only has k non-zero elements, so we only need to transmit $O(k)$ bytes. We will add a remark to make this clear. 5. *"TODO"* - Thanks!
6. *"extends to methods like Adam?"* - This is a good point. The Adam update uses the ℓ_2 norm of the sum (or linear combination) of gradients in the update, and count sketch can be used to approximate the ℓ_2 norm. Therefore we can, in principle, mimic the Adam update. We do not know if this approximation is good enough to preserve Adam's theoretical properties, or if it would work well in practice.
7. *Regarding momentum?* - Sorry for the poor terminology – all experiments use momentum.
8. *"..why momentum factor masking 'interferes with momentum'"* - in line 11 of Algorithm 3, we zero out the coordinates of each u_i that were updated (this is momentum factor masking). In the limit of $k \to d$, each coordinate is updated every iteration, so there is effectively no momentum.
9.*"communication reduction at the expense of model accuracy"* - To be clear, we get no drop in performance with 40x compression for the translation task. For CIFAR-10, we can make up the accuracy drop by training longer. For example, training with 17x compression for the usual number of iterations gives 92.5% validation accuracy. Training with 50% more iterations (reducing to 11x overall compression) restores accuracy to 94%. We expect similar results to hold for the translation task at compression rates higher than 40x.
10. *"compare empirically against others"* - We outperform many popular methods. For example, the "Local Top-k" method in Figure 4 is very similar to deep gradient compression (Lin+2017), except we do not clip gradients, and we warm up the learning rate instead of the sparsity. Local Top-k does about as well as 9x-compression Sketched SGD, but it achieves only $\sim 2x$ compression for large enough W. We also implemented and compared to signSGD (arXiv:1810.05291), which gets 92.5% validation and 32x compression. When we quantize all communication in Sketched SGD to 16 bits, we achieve 92.5% accuracy with 34x compression ($r = 3$, $c = 100,000$, $P = 8$). For the translation task, we already achieve 40x compression with no loss in accuracy, which is better than signSGD's 32x, and which we expect to double to 80x without loss of accuracy by adding 16-bit quantization. TernGrad (arxiv:1705.07878) is another popular method, but achieves hardly any compression of the back-communication from parameter server to workers for the large models we consider with $d \approx 2^{26}$ – see TernGrad pg. 3.