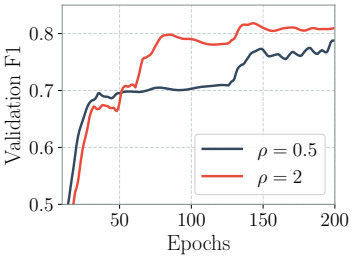
We are grateful for the useful and constructive comments from all anonymous reviewers. **(Large dataset)** As suggested by reviewers, we report the result of LazyGCN on Amazon dataset (1.5m nodes) in Table A1.

Table A1: Comparison of the test F1 score and time (addition to Table 2)

		NodeWise	**+LG**	LayerWise	**+LG**	SubGraph	**+LG**
Amazon	Testing F1	77.29%	76.99%	77.23%	77.10%	77.25%	77.12%
	Time (s)	5092.8	**463.9**	571.4	**206.8**	385.2	**198.4**

Reviewer 1 (Train vs inference) We thank the reviewer for the thoughtful comments. In this paper, we aim to reduce training time for sampling-based GCN, which has been widely used to scale GCN training on extremely large graphs. Upon carefully reading of the suggested paper by reviewer, we note that although this paper also proposes a new method to accelerate GCN training and inference, it requires a specially designed GCN structure, while our method can be jointly used with any sampling-based GCN models, such as GraphSAGE, FastGCN, and GAT. **(Sample / transfer)** We appreciate bringing up this matter. It is worth mentioning that sampling time increases significantly as the graph size grows, while the transfer and computation time on GPUs remain the same, given the limited GPU capacity. In fact, there are a few key factors that can degrade the performance of sampling for GCN: (a) Unlike GPU that only process a fraction of data, CPUs need to work on the entire graph, which imposes significant overheads for both computation (random memory accesses for traversing the large data structure) and storage (limited main memory); (b) When the size of the graph is large, the graph may not be fully loaded into the main memory (e.g., RAM), and it must be stored in secondary storage (disk). Doing so will incur extra time to transfer data from secondary storage to main memory. Besides, as shown in Figure 1, the GPU is stalled after mini-batch 2 due to the sampling process of a mini-batch 3. Although in this case, the GPU idle time seems negligible compared to the overhead of transfer, when sampling time increases (for a fixed number of CPUs), the idle time will further slow down the training. We will make sure to highlight this overhead in the revision. **(Related work)** To the best of our knowledge, we are the first paper working on accelerating the GCN training by reducing the sampling and transfer time. Besides, similar problems faced in other fields are discussed in lines 75-78.

Reviewer 2 (Why $\rho > 1$) The intuition comes from *exploration* and *exploiting* trade-off in standard convex and non-convex optimization analysis. At the beginning of training, our solution $\boldsymbol{\theta}_t$ is far away from stationary point $\boldsymbol{\theta}^\star$ and the gradient $\|\nabla F(\boldsymbol{\theta}_t)\|$ is large. At this point in time, more recycling on a sampled mini-batch might cause an overfit on that mini-batch, while more fresh samples (less recycling) enable us to find the right direction toward optimal solution (exploring). As the optimization proceed, the gradient vanishes and $\|\nabla F(\boldsymbol{\theta}_t)\|$ becomes small. As a result, the possibility of overfitting is much smaller, which allows for more recycling (exploiting). Besides, as suggested by reviewer, we demonstrate the effect of different ρ on the convergence of LayerWise method in the figure on the right hand side. It can be seen that smaller ρ requires more time to recover from overfitted model. **(Multi-level version)** We thank reviewer for the careful reading. Indeed, the variance at the ℓth layer will affect the variance of $\{\ell+1, \dots, L\}$ layers, where L is the number of layers. We will provide the analysis for the multi-level case in the subsequent version. However, we want to point out that the single layer GCN can be formulated as a two level optimization problem (L-layer GCN can be formulated as $L+1$ level optimization problem) where the variance at the 1st level already influence the variance at the 2nd level. **(Effect of ρ on upper bound)** We note that the total number of iterations is constant, hence larger ρ leads to smaller K and does not affect the bound. **(Epoch size)** Note that the size of the kth epoch is $\rho^k R$ iterations in Algorithm 1, which is increasing during training, while the epoch size is fixed in vanilla GCN. We illustrate the validation scores for every 10 iterations for both settings to make the figure readable. **(Figure 2(c) clarification)** In this figure we want to show LazyGCN can increase *the fraction of time on computing* during training. The figure is normalized (divided) by total wall-clock time to show the proportion of each phase and the total wall-clock time are reported in Table 2. **(Notation $\mathcal{B}$)** Eq.7 shows the stochastic gradient computed on mini-batch $\mathcal{B}$ in LazyGCN is close to the stochastic gradient of GCN without node sampling. Therefore, the subscript $\mathcal{B}$ shouldn't be ignored. **(Related works)** We thank reviewer for the advise. Our goal is to develop a general yet effective framework that can be integrated with any sampling strategy to substantially improve the training time. Our experiments are conducted under the same setups (e.g., GCN architecture and hyper-parameters) as the backbone method. Notice that both [A] and [B] can be regarded as an incremental method of GraphSAGE with a similar sampling strategy and we would surely love to include the discussions on them in the subsequent version.



Reviewer 3 and 4 We thank both reviewers for the suggestions. Please refer to the additional result on the Amazon dataset at the top of this page (**Large dataset**). We would also like to further evaluate LazyGCN on newly released datasets (OGB) and other settings in the subsequent version.