We thank the reviewers for their detailed comments. We are glad to see a generally positive assessment of our work. The main aim of our work is to develop reversible graph neural network models, called Graph Normalizing Flows (GNFs) which can be used for both supervised learning and unsupervised learning. On supervised tasks, we show that the GNF model exhibits performance comparable to a regular GNN model while providing the advantage of a significantly lower memory footprint. In the unsupervised setting, we develop a permutation-invariant generative model for generating entire graphs in parallel, and also demonstrate the applicability of the model for more accurate density estimation.

We agree with the reviewers that scaling the model to larger graphs is an important problem and are actively working in this area. We will report larger-scale results in the final draft. As the reviewers highlight, the model is a novel approach to this challenging problem and possesses many interesting and useful properties. We believe it will be of great interest to the NeurIPS community. Below, we address specific reviewer comments.

R1, R3: Memory footprint We first provide a more rigorous theoretical derivation for the memory footprint and then provide some quantitative results. Let us assume that the node feature dimension is d, and the maximum number of nodes in a graph is N. Let us assume weights (parameters) of the message passing function is a matrix of size W. For simplicity, assume a parameter-free aggregation function that sums over messages from neighbouring nodes. Finally, assume that the final classifier weights are C in size. Suppose we run K message passing steps. Total memory that needs to be allocated for a run of GNN (ignoring gradients for now; gradients will scale by a similar factor) is $W + C + K \times N \times d$ (= memory allotted to weights + intermediate graph-sized tensors generated + adjacency matrix). For a GNF, the total memory is $W + C + N \times d$. Note the *lack* of multiplicative dependence on the number of message passing steps in the latter term.

As a quantitative example, consider a semi-supervised classification task on the Pubmed network ($N = 19717, d = 500$). We assume that the message passing function for a GNN is as follows: $\text{FC}(500) \rightarrow \text{ReLU}() \rightarrow \text{FC}(750) \rightarrow \text{ReLU}() \rightarrow \text{FC}(500)$. Each of the functions $\text{F}_1(\cdot)$ and $\text{F}_2(\cdot)$ (please see Figure 1 in the paper for notation) in the corresponding GNF have the following architecture: $\text{FC}(250) \rightarrow \text{FC}(750) \rightarrow \text{FC}(250)$. We can compute the total memory allocated to weights/parameters: $W_{GNN} = 500 \times 750 + 750 \times 500, W_{GNF} = 2 \times (250 \times 750 + 750 \times 250)$. We perform $K = 5$ message passing steps for Pubmed. So, the amount of memory allocated to intermediate tensors in a GNN is $19717 \times 500 \times 5 + 19717 \times 750 \times 5$, and correspondingly for a GNF is 19717×500. Summing up, the overall memory requirements are: GNN = **945.9 M** and GNF = **80.2 M**. Hence, in this case, GNFs are at least $\geq$ **10**$\times$ memory efficient than GNNs. Further, we use self-attention in our experiments, which scales according to $\mathcal{O}(N^2)$. GNNs will store attention affinity matrices for each message passing step. In this case, a similar argument can show that this causes a difference of **11G** memory. When using 12G GPU machines, this difference is significant. We will add in a table of memory consumption on all the datasets reported in the paper in the final version.

R3, R5: results are marginally better than the baselines (QM9). The baselines are also not the SOTA techniques; no significant performance boost on any of them; The improvements on the supervised tasks are very marginal compared to vanilla GNN. The main aim of the GNF model in the supervised setting is to reduce the memory consumption significantly while providing comparable performance to their GNN counterparts. We agree that the QM9 results are not SOTA, however, we made sure that the comparison performed is fair – the GNN and the GNF architectures were identical with only reversibility being the exception. We will add a comparison between GNF and SOTA GNN techniques (like Graph Attention Networks and any other that the reviewer suggests) in the final version.

R2: Have you considered balancing the loss for positive and negative edges? This is a good suggestion. We did investigate the error distribution (false positives and false negatives) and found it to be roughly evenly distributed. In cases where the imbalance does hurt though, this idea could certainly be helpful.

R3: Also, what is the speed/complexity of GNFs compared to GNNs? GNFs require 1 extra forward pass during backpropagation as the forward tensors need to be computed for gradient propagation. So, the overall run of a GNF consists of 2 forward passes for each backward pass as compared to 1 forward pass and 1 backward pass for a GNN. We note however, this (relatively cheap) 1 additional forward pass comes at a huge memory benefit.

R5: Over simplified dataset on Density estimation which is an important application. No experiments on other density estimation benchmarks. In section 5.1, we demonstrate the effectiveness of GNFs for *structured density estimation*. The goal of structured density estimation is to model densities of a set of inputs. In our experiment (Figure 2), we model densities of a set of four points – one drawn from each Gaussian. Figure 2 shows that modeling densities of points together can outperform modeling per-example (e.g., iid) densities, using a RealNVP, independently. We are not aware of any standard benchmark for this task however if the reviewer has any suggestions, then we can certainly perform experiments on more complicated datasets.