We thank all the reviewers for their constructive comments. We first explain the update of our model which resolves some of R2's questions. Then we address the common concern on the large scale graph generation followed by individual responses. We will release our PyTorch implementation upon acceptance.

Model Simplification & Extension: After submission, we tried removing the RNN, thus simplifying our model to a single GNN. With more layers and slightly more training epochs, a single GNN performs similarly to the model with an RNN. Moreover, we extended the output distribution from a single Bernoulli distribution (Eq.7 in the paper) to a mixture of Bernoullis, which can further model the dependency between the entries in a block without sacrificing the speed, and achieves state-of-the-art performance on most metrics of Protein dataset (*c.f.* Table 1).

Protein Dataset	Deg.	Clus.	Orbit	Spec.
GraphVAE	$6.12e^{-2}$	0.17	0.82	$9.28e^{-2}$
GraphRNN	$\mathbf{5.51e^{-3}}$	0.15	0.40	$2.56e^{-2}$
GRAN-50	$1.08e^{-2}$	**0.08**	**0.30**	$\mathbf{8.97e^{-3}}$

SBM Graphs	Deg.	Clus.	Orbit
$G(5000, 0.05)$	0.78	1.93	0.37
$G(5000, 0.1)$	0.78	1.83	0.37
GRAN-20	**0.54**	**0.99**	**0.34**

Table 1: GRAN-X: our model (RNN removed) with a mixture of X Bernoulli distributions. $G(n, p)$: Erdos Renyi model with number of nodes n and edge probability p.

Large Scale Graph Generation [R2, R3]: We experimented with random graphs from stochastic block models (SBM) to show that our model is able to generate large graphs. In particular, we generated 10 random graphs from an SBM with 3 communities, 5000 nodes and inter/intra-community probabilities $[[0.2, 0.001, 0.002], [0.001, 0.15, 0.004], [0.002, 0.004, 0.1]]$. We randomly choose 8 graphs for training and test our model on the remaining 2 graphs. The performances of our model and several Erdos-Renyi random graph baselines are shown in Table 1. The run-time of generating 3 graphs (5K-nodes) in parallel on a GeForce 1080Ti GPU is around 279s. We experimented with the public implementation of GraphRNN which runs out-of-memory under this graph size. We are investigating the suggested real-world graphs and will include the results in the final version.

How to determine graph size [R2, R5]: We set a maximum graph size which determines when the GNN stops. Then we sample the graph size from the empirical distribution of graph sizes (collected from the training set). Finally, we crop the generated adjacency matrix from top-left using this sampled graph size. Although learning to predict the graph size is arguably better, we found this simple sampling strategy works well in practice.

To R1:

- Evaluation. We first compute the statistics, *e.g.*, degree distribution, of the generated and observed graphs respectively. Then we measure the similarity between them using the maximum mean discrepancy (MMD) with a Gaussian kernel using total variation distance. For spectra, *i.e.*, eigenvalue distributions, we discretize the range and treat it as a histogram. In general, it is hard to measure differences using a single metric, as each statistic is just one particular summarization of the graph. Low MMD scores are necessary but not sufficient for two graphs to be similar. In our experience, small differences of MMD scores in degree distribution and spectrum, *e.g.*, 0.01 vs. 0.02, seem not to imply a significant difference in the quality of the visualized graphs whereas clustering coefficients and orbits correlate better with the visual quality. We will discuss this point further, and show examples along with the metrics.

To R2:

- How Eq. (11) is trained? $q(\pi|G)$ is a 2-layer GCN which takes rows in the adjacency matrix as the input node representation. We also tried random node features and obtained similar performance. There is no ground truth for training this network; the only training signal comes from maximizing the ELBO.

- Maximum graph size. Please refer to the section above. **- a in Eq.(5).** They are scalars and normalized by the sigmoid. **- Remove RNN.** We have since done this; see above. **- Overall Training.** It is trained by teacher forcing. All subgraphs are trained in parallel as in a pixel CNN. **- Model Size.** On the protein dataset, the size of the best-performing GraphVAE vs. GraphRNN vs. GRAN-20 is roughly 128MB vs. 2MB vs. 4MB. We will include it in the final version.

To R3: Please see our response above on large scale graph generation.

To R5:

- Multiple splits. Thanks for the suggestion! We will cite that paper and add the results in the final version. **- Optimization of the GNN.** We used the Adam optimizer with learning rate 1.0e-2 and decay by 0.3 at 100, 500, and 1000 epochs. The norm of the gradient is clipped at 5. We will discuss it in the final version. **- Long range dependency.** We believe our model with the GNN and less number of generation steps do have a positive impact on this. But it is not easy to demonstrate this explicitly through an experiment. **- Graph size.** Please refer to the graph size section above.