We would like to thank each of the reviewers for reading our manuscript and providing very useful feedback. For minor comments such as typos, missing citations, and choices of words/notation, we have fixed them. The reviewers raised a few points that warrant more discussion than we are able to fit in the manuscript with the space constraints. Therefore, we have expanded the supplementary section in the revised version. Below, we address the key points in detail.

R3: Runtime and scalability. Here we provide the runtime of vGraph and the two fastest overlapping community detection baselines on DBLP-full: vGraph trained for 5000 iterations with batch size 1000 (277s), BigCLAM (193s), CESNA (1191s). Note that it is not a fair comparison as our code is in Python whereas the other codes are implemented in C++. Below we further show the time complexity of vGraph. Sampling an edge takes constant time, thus calculating Eq. 4 takes $O(d(M+1))$ time, where M is the number of negative samples and d is the dimension of embeddings (the node embeddings and community embeddings have the same dimension). To calculate Eq. 6, it takes $O(dK)$ time where K is the number of communities. Thus, an iteration with one sample takes $O(\max(dM, dK))$ time. In practice the number of updates required is proportional to the number of edges $O(|\mathcal{E}|)$, thus the overall time complexity of vGraph is $O(|\mathcal{E}|d\max(M,K))$. The fact that (1) it scales linearly to the number of edges and (2) it employs negative sampling combined with batch-wise stochastic optimization makes vGraph scalable. We will add this analysis in the revised version. **R3: Regarding the aggregation over edges** We aggregate over all edges by taking summation of the ELBO bound on each edge.

R2: Benefits of vGraph. From Tables 2, 3, and 4 we can see that vGraph outperforms methods that perform community detection or node representation individually since vGraph integrates the two tasks, which are beneficial to each other. Moreover, vGraph is efficient and scalable compared to classical community detection methods.

R1&R3: Regarding the design of the two sets of node embeddings. R1: Recall our generative process: (1) we first draw a community assignment $z \sim p(z|w)$ representing the social context of node w, (2) then based on z, we generate a neighbor of w, (c can be referred to as the "context" of w) $c \sim p(c|z)$. The first set of node embeddings is used in step (1) and the second set of node embeddings is used for step (2) in the generation process. A similar concept is used in existing node representation learning methods (e.g. DeepWalk, LINE, node2vec) and matrix factorization methods where they use two different sets of node embeddings. R3: Note that in vGraph, the two set of embeddings are tied together by community embeddings and thus capture similar information. For the final embedding, we take the first set of node embeddings (ϕ). In fact, we can also share parameters for two embeddings (that is, $\phi = \varphi$) and it yielded similar performance. We have updated the manuscript to make this more clear.

R1&R3: Design of smoothness regularization. R1: For the distribution distance d, we did experiment with divergence based distance metrics and found they did not yield much difference. Thus, we used squared difference as in [25] for the sake of simplicity. R3: Indeed, by designing smoothness regularization the way it is in the paper, we are implicitly considering communities based on assortativity. We agree that it is worth exploring different forms of smoothing functions that possibly favor detecting different kinds of communities. For now, we left this as future work.

R3: Experiment settings and design. We agree that there are many efficient approaches to community detection, as you pointed out in the review, and covering all of them is difficult. **(1) About baselines.** In fact, our experiments are designed to demonstrate that the vGraph framework enables community detection and node representation learning to benefit one other, not to prove that it outperforms all existing studies. Therefore, we decided to choose certain representative methods (i.e., matrix factorization-based methods, generative models, and K-Means after node embeddings) which help validate this point. We will discuss more studies in the revised draft. **(2) About choosing K.** In practice, when the true K is not given, we can still choose K according to the performance on validation set (as in [14,36]). However, existing studies typically assume that the oracle K is given in experiments [4,18,25,30,32]. We follow the same experiment setting. Also, those parametric models compare only with other parametric methods, not non-parametric models such as Louvain. In practice, we can still compare with non-parametric methods but we have to make sure comparisons are made under the same model complexity (i.e. the same number of communities). **(3) About cases with many small communities.** Since communities are determined based on node embeddings, the algorithm works regardless of the number of communities. **(4) On evaluation on ground truth communities.** We agree that there may be multiple equally correct ground truths in practice. However, all the datasets we consider only have one "ground-truth" label. Thus, testing community detection algorithms on the given ground-truth is the best we can do and it is still the most widely adopted way to evaluate community detection [4,14,18,20,36,37]. Furthermore, we use modularity as one of the evaluation metrics in our experiments (Table 3) and modularity does not depend on ground truth communities. **(5) In the classification experiment, what classifier are you using on the node-embeddings?** We employ an one-vs-rest logistic regression classifier using the commonly used LIBLINEAR package.

R3: Can you modify vGraph to incorporate nodal attributes? vGraph is more of a principled framework to integrate community detection and node representation learning, where it is flexible to use different kinds of encoders for learning node representations. To incorporate node attributes, we can simply use graph neural networks (e.g., Graph Convolutional Networks) as the node encoder.