We would like to sincerely thank all our reviewers for their valuable feedback and insightful comments. We are committed to doing our best in addressing their concerns and suggestions in the final version of the paper. We would like to reiterate that apart from making the suggested clarifying revisions to our paper, we will release all of the datasets, source code and hyper-parameter configurations used to ease the reproducibility of our work.

Formal meaning of the word "family" in Section 4.2 (Review #1): In experiments on generalizability, we've used the term "family" in lieu of a more formal choice: "distribution". More precisely, we're interested in evaluating whether a placement policy trained with graphs sampled from a particular distribution can generalize to different, unseen graphs sampled from the same distribution. In our evaluation, we pick these distributions to correspond to neural network architectures designed for a specific task, such as image recognition (*cifar-10*), language modeling (*ptb*) and language translation (*nmt*). Appendix A.3 describes how we generated neural networks for each of these tasks using the ENAS method for *cifar-10* and *ptb*, and variations of the NMT model for *nmt*. We will clarify this point in the revision.

Choice of REINFORCE algorithm for RL optimization (Review #2): We've chosen REINFORCE for RL optimization in order to establish a fair comparison to the prior work, a majority of which uses it as well (Mirhoseini *et al.* (2017), Mirhoseini *et al.* (2018) and Paliwal *et al.* (2019)). Furthermore, we believe that our core contributions are orthogonal to any specific choice of the optimization algorithm used. However, we do agree that it would be interesting to verify if our results improve with the use of a better algorithm that plays a more significant role in variance reduction. We leave this investigation for future work.

Graph Neural Network architecture (Review #2): Placeto's policy is computed by running two separate, independent graph neural networks concurrently and concatenating node embeddings obtained from them together to form embeddings that get used for placement decisions. The first graph neural network iteratively aggregates messages for k steps "bottom-up" from the children nodes using the functions f and g as follows: $\mathbf{x}_v^{(i+1)} \leftarrow g(\sum_{u \in \xi(v)} f(\mathbf{x}_u^{(i)}))$, where $\xi(v)$ are the children of node v. The second graph neural network performs a similar update "top-down" from the parent nodes instead. k is a tunable hyperparameter, which can be set to sweep the entire graph or to integrate information from a local neighborhood of each node that is sufficient for making good placement decisions. In our experiments, we found that sweeping the entire graph is computationally expensive and provides little benefit compared to $k = 8$. Hence we use $k = 8$ in all of our experiments. We will revise Section 2.2 and Figure 3 to clarify this point. Regarding the use of feed-forward neural networks in the message passing operations of the GNN, our architecture is similar to past work (*e.g.,* Mao *et al.* (2018), Battaglia *et al.* (2018)).

Additional details: We use a graph embedding size of $E = 8$. All our feed-forward networks have a single hidden layer. We will add these details to the appendix section A.7 and additionally, release a source file with all of of our hyper-parameter configurations along with the source code for easy reproducibility of our results.

Ablation study of bidirectional GNN (Review #2): We agree that it would make for an interesting addition to compare our bi-directional approach to a more standard one where messages from all the neighbors (parents and children nodes) are aggregated jointly at each step of the message passing update operation. We plan on performing this ablation study and including its results in the final version of our paper.

Clarification on iterative placement improvement vs. RNN (Review #2): While the RNN approach could indeed condition placement of future nodes on those chosen for prior nodes, our formulation makes this conditioning explicit in the state space of the MDP, since the state includes the current placements as a feature associated with each node. By contrast, prior RNN-based architectures (e.g., the Encoder/Decoder approach in Mirhoseini et al.) must realize such conditioning implicitly via the hidden state of the RNN. As the reviewer observes, making this conditioning explicit also enables us to perform multiple epochs of placement improvements. Although we failed to mention this, we actually do perform multiple epochs of iterative placement improvement in our experiments. More specifically, we found that performing 2 rounds of epochs strikes a good balance between performance and speed. We will revise Section 2.1 to explain this point precisely. In particular, we will modify the MDP description to state that the episode ends in $n|V|$ steps, where n is the number of epochs.

Stopping criterion used for Section 4.1 (Review #2): The stopping criterion used to determine the training time column in Table 1 is the numerical convergence of reward signal over a long enough measurement window. Precisely, the reward value doesn't change more than 2% in a measurement window of 1000 episodes.

More succinct choice of reward formulation (Review #2): Thank you for the suggestion! We will amend the reward section in the Appendix A.7 to reflect this more succinct $ReLU$-based expression.

Node ordering used for training (Review #3): We pick an arbitrary topological ordering (in which parents nodes appear before their children) by default. This ordering is indeed not learned. However, as we show in Section 4.3, the order does not impact Placeto, and can even be completely random (not topological) without having a significant impact on generalizability of the learned policy for Placeto.