We thank all reviewers for their encouraging and constructive feedback.

Rev#1: We thank the reviewer for their suggestion to tease apart the impact of architectural decisions, have a clearer elaboration of the fusion task, and include full neural network model formulation. We will include additional details and ablation studies in the revised version to further clarify each of these aspects.

[Fusion priority] As mentioned in Sec 3 of the paper, each node is associated with a fusion priority score (ranging from 1 to F, where F is a hyperparameter). The scores determine the order in which nodes are fused.

[Better figure] The reviewer is correct that each Transformer layer in Fig.3 only outputs a feature embedding that is used by the next layer, and only the final layer computes actions. In Eq.2, the actions are produced by the final layer in Fig.3. However, the temporal dependencies between actions in multiple RL steps are carried out by state embedding such that the state of the previous step is used as an input to the network in the next step. Fig.3 is an illustration of a single task, while Fig.4 demonstrates the extension of the same framework for multiple tasks. We add a few more Transformer layers with residual connections as task heads. The yellow part in Fig.4 is the same as Fig.3.

[Why not use the first layer of the TRF to replace the modulation layer?] Having a separate modulation layer provides additional parameterization to the network and hence better orchestration of the training of the parameters. It empirically results in better generalization.

Rev#2: We thank the reviewer's insightful feedback on providing more details and potential open sourcing for reproducibility. We will add more detailed explanations about GNN's limitations on tracking global node dependencies.

[Reproducibility] We made an extensive effort to document the details of our network architecture, the input graphs, and other hyperparameters in Sec. 4 and Supp.A. We plan to incorporate further details to enhance reproducibility and open source the GO framework along with the performance model. We would also like to emphasize that GO, as a general algorithm, can be applied to problems at different layers of the compilation stack, even on different platforms. We have successfully applied GO to two other graph optimization problems at different stages in the compiler stack – tensor layout optimization and operator fusion in XLA (as opposed to at the TF level discussed in this paper).

[Can GO be trained online?] Yes, GO can be trained in an on-line scenario similar to Decima. Specifically, GO-one is quite similar to an online training method where state transitions and rewards can be collected by interacting with the compiler. However, as pointed out, an online RL can make poor decisions in early stages of training and the quality of training is subjective to input data distribution. This is particular challenging for a compilation problem, where a program needs to be compiled within a short time. One possible approach is to pre-train the model like GO-finetune, then apply online training to continuously improve the policy.

Rev#3: We appreciate the reviewer's detailed feedback and will address the questions in the final version.

[Reproducibility and validation] Please see our response to Rev#2. The accuracy of the performance model has been validated against true runtime measurements (on actual hardware) on several industry standard models, including MLP, CNN, RNN, LSTM, Transformer, BERT, etc. We will provide more validation results in the final paper.

[Table1: is there a methodology for these yes/no distinctions?] We empirically observed these distinctions. For example, a vanilla Transformer network will be OOM for problems larger than 10k nodes, and a vanilla LSTM cannot generate placement decisions for input sequence longer than a few hundreds. Ablation study in Supp.C.1 addresses some of the distinctions.

[Source for the claim the workloads are realistic] We used important workloads that are widely used in real applications or are incorporated in industry standard benchmarks like MLPerf. These include ResNet, InceptionNet, WaveNet, Transformer-XL, AmoebaNet, NMT, and RNNLM.

[SA convergence time] SA takes around 24 hours to converge in our evaluated tasks. For a smaller graph, GO takes less than an hour to converge. For a large graph over 10k nodes, GO can take 1-4 hours to converge.

[Better figure] We acknowledge that Figure 3 can be improved, and we will incorporate the suggestions from the reviewer to the final version.

[More details of the performance model] Exact features used in node embedding (as presented in Fig.3 dashed box): tensor shape, op type, and adjacency matrix. Fusion is optimized using a priority-based node traversal. Please also see reponse to Rev#1 [Fusion priority].

[Is this function $f^{(l+1)}$ different each iteration?] Yes. ℓ indexes the layer of the graph neural net and different layers of the graph neural net have different f.

Rev#4: As mentioned in response to Rev#2, we plan to open source the GO framework along with the performance model, and will include details about this in the final version.