Thank you for the thorough reviews and insightful comments.

Clarifications We first address some of the smaller points raised by the reviewers. R1, we clarify that the loss function (line 287) is indeed a cross-entropy loss on the 1000 possible values for v_0 % 1000. R1, the accuracy in Table 2 is (just as you surmised) the percent of examples in the test set that the model produces the exact correct result for. Recall that the test set has programs with lengths 20 through 100, which places them outside of the train distribution. R3, we clarify that the "soft instruction pointer" is the novel attention mechanism that the paper draws its title from. We discuss it in detail in Sections 4.2-4.4 and Section 5. We have revised the paper to clarify each of these points. R2, we like your suggestion of adding an overview diagram and will gladly make this addition for the camera-ready if accepted.

Why GGNN? And a new baseline: R-GAT R3, the primary reason we elect in the paper to compare against the GGNN model rather than e.g. GAT or GraphSage is that GGNN is a more common model choice for program understanding tasks [1, 4–8]. Additionally Table 3 of [2] compares variants of common message passing GNN architectures (GGNN, GCN, and GAT) on the VariableMisuse task and finds the degree of variation in performance to be considerably smaller than that between the GGNN and IPA-GNN on learning to execute. So, we do not think that the comparatively small differences in these GNN architectures will yield significant differences on the learning to execute tasks.

Nevertheless, we take R1 and R3's suggestion to introduce a new baseline, R-GAT [3] (non-relational GAT lacks edge-types and so cannot distinguish between true and false branches of if statements). Preliminary results suggest similar performance to the GGNN. We will incorporate the R-GAT baseline results into our paper if accepted.

"NoControl" model equation R1 asked about our claim that "GGNN = NoExecute and NoControl". Indeed, we made a typographic error in Table 1, and we thank R1 for catching our mistake. The correct equation for $a^{(2)}_{t,n,n'}$ for the NoControl model is simply $a^{(2)}_{t,n,n'} = a^{(1)}_{t,n}$. Please note the difference between this and what is printed in Table 1 in the paper. We checked and the error is purely typographic, not an error with the experiments themselves. We have now corrected this, and we hope this resolves your concern. With this fix in place, it should be apparent that the NoControl and NoExecute models interpolate between the IPA-GNN and GGNN model. When both the NoControl and NoExecute changes are simultaneously applied to the IPA-GNN model definition, the result is the GGNN model. R2, we hope this correction also addresses your request for improved evaluation of the individual components of the IPA-GNN. We stress for R2 that the primary purpose of the NoControl and NoExecute models is to determine the significance of the individual components of the IPA-GNN model. These models form an ablation study of sorts, but rather than *ablating* the individual components of the IPA-GNN model (which would render it useless) we replace them with the analogous component from the GGNN model. So we think of this more as an "interpolation" study than an "ablation" study, but it serves the same purpose.

Significance of improvement We note a disagreement between the reviewers about the significance of the improvement our architecture contributes ($26.7\% \to 43.8\%$ and $17.2\% \to 28.3\%$ out-of-distribution accuracy on full and partial program execution respectively). R3 notes that this improvement is not strong enough. We think this improvement is more than enough to warrant publication, and respectfully request that R3 reconsider their stance. We hope the reviewers get to discuss this point in their post author response discussions.

References

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs, 2017.

[2] Marc Brockschmidt. {GNN}-fi{lm}: Graph neural networks with feature-wise linear modulation, 2020. URL `https://openreview.net/forum?id=HJe4Cp4KwH`.

[3] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y. Hammerla. Relational graph attention networks, 2019. URL `https://openreview.net/forum?id=Bklzkh0qFm`.

[4] Mingzhe Li, Jianrui Pei, Jin He, Kevin Song, Frank Che, Yongfeng Huang, and Chitai Wang. Using ggnn to recommend log statement level, 2019.

[5] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2015.

[6] Mingming Lu, Dingwu Tan, Naixue Xiong, Zailiang Chen, and Haifeng Li. Program classification using gated graph attention neural network for online programming service, 2019.

[7] Jessica Schrouff, Kai Wohlfahrt, Bruno Marnette, and Liam Atkinson. Inferring javascript types using graph neural networks, 2019.

[8] Sahil Suneja, Yunhui Zheng, Yufan Zhuang, Jim Laredo, and Alessandro Morari. Learning to map source code to software vulnerability using code-as-a-graph, 2020.