Thank you to all reviewers for the very careful feedback. We respond here, and **we will update the paper accordingly**!

(**Experiments**) We evaluated additional models, providing further empirical evidence on the PGNs' benefits: (**SupGNN**) baseline GNN whose latents have been supervised to be predictable of masks and pointers (without explicitly using this information); (**PGN-Asym**) PGN without symmetrising pointers; (**PGN-MO**) PGN where only masks are supervised (and pointers are not directly optimised); and (**DGM**) [19], a recent strong baseline method on latent graph inference. Here we provide results on the **hardest** test set (LCT; $n = 100$; ops $= 150$); results on other sets mirror these findings.

GNN	**SupGNN**	**DGM**	**PGN-MO**	**PGN-Asym**	**PGN**
$0.401\pm.123$	$0.541\pm.059$	$0.524\pm.104$	$0.558\pm.022$	$0.561\pm.035$	$\mathbf{0.616}\pm.009$

(**R1 / R2 / R3**) Our work evaluates the extent to which existing neural networks are potent reasoning systems, and the minimal ways (e.g. inductive biases / training regimes) to strengthen their reasoning capability. Hence our aim is not to outperform classical algorithms, but make their concepts accessible to neural networks. PGNs enable reasoning over edges not provided in the input, simplifying execution of any algorithm requiring a pointer-based data structure. PGNs can find direct practical usage if, e.g., they are pre-trained on known algorithms and then deployed on tasks which may require similar kinds of reasoning (with encoders/decoders "casting" the new problem into the PGN's latent space).

(**R1**) Inspired by your review, we scaled up the LCT test set to ($n = 200, \text{ops} = 300$) where the PGN ($0.636\pm.009$) catches up to Oracle-Ptr ($0.619\pm.043$). We hope that this illustrates not only the robustness of PGNs to larger test sets, but also provides a quantitative substantiation to our claim about ground-truth LCTs not having favourable diameters.

(**R1**) Our contribution is *both* in the inductive biases (e.g. pointers/masks) proposed and the provided supervision. To strengthen this claim, we compare against **SupGNN** in experiments above. Supervising provides a clear boost in performance, but is still insufficient to outperform PGNs, showing the importance of the architectural choices as well.

(**R2 / R1**) To strengthen our empirical contribution on learning latent graphs, we also compare against the recently proposed **DGM** [19] loss function in the experiments above, finding PGNs extrapolate much better to larger test sets.

(**R2 / R4**) We added experiments on asymmetric pointers (**PGN-Asym**), with a clear drop in performance on the LCT task. Corresponding to R4's comment, we find that while symmetrising the pointers is not strictly necessary, empirically it is helpful because it makes the inferred graph slightly less sparse, enabling us to rectify for the case where the pointer mechanism makes mistakes without sacrificing sparsity. In this sense, a more "global" view of each node is desirable.

(**R3**) Our work extends [42] by the addition of the **pointer / masking** inductive biases, allowing one to more efficiently simulate algorithms for which the input graph does not correspond to the actual links used for reasoning, or cases where an input graph is not provided at all. As we acknowledge, while conditional masking had been introduced in [50], here we utilise masking to update node state rather than exclude it from the input—which can reduce information loss.

(**R3**) We now compare against **PGN-MO** in the experiments above, which supervises on **m**asks **o**nly and does not directly optimise pointers. The results show that we can relax most of the supervision constraints and still outperform baseline approaches (such as GNN and DGM), while for full returns an explicit pointer-based loss is required.

(**R3**) Please note that PGN-Ptrs requires *two passes*: first training a PGN, then re-using and fixing its pointers on a second training pass. Allowing for dynamic adjusting of pointers works better on complex algorithms with less margin for error, such as the LCT setup. If the PGN mechanism makes a mistake at any epoch, PGN training can meaningfully recover from this while PGN-Ptrs don't have this flexibility. For DSU, path compression yields very shallow data structures that are modellable in many equivalent ways, hence such errors are less punishing for PGN-Ptrs.

(**R3**) Thank you for remarking the correctness of our "expressivity" claim—we will amend as you suggested!

(**R3**) Various answers: F_1 measures performance across the (binary) query responses $y^{(t)}$, handling class imbalance. As mistakes are measured on $y^{(t)}$, errors at step t do not imply errors at step $t + 1$. By parallelisable, we imply that they allow for using GPUs more effectively (e.g. the rollout vs. ground-truth in Fig. 4). 1-NN refers to 1-nearest neighbours, for which efficient algorithms exist and we don't need to compute all $O(n^2)$ α_{ij} values. $\|$ is concatenation.

(**R4**) We agree that multiple GNN steps may be theoretically required to reconcile our work with teacher forcing the data structure. We found it sufficient to consider only one-step here, but as tasks get more complex—especially when compression is no longer applicable—dynamic number of steps (e.g. function of dataset size) is likely to be appropriate.

(**R4**) Thank you for suggesting the GAT experiment! While it seemed to compare favourably to GNN/DeepSets on the (more compressible) DSU task, the unrestricted GAT model failed to get off the ground at all for the LCT task.

(**R4**) Thank you for the minor comments, which we will incorporate! Especially for pointing out the work of Goel *et al.*, which allows us to report a tighter analysis on time complexity (as we effectively use randomised linking-by-index).