Recurrent Aggregators in Neural Algorithmic Reasoning

Kaijia Xu
University of Cambridge
kx233@cantab.ac.uk

Petar Veličković
Google DeepMind
petarv@google.com

Abstract

Neural algorithmic reasoning (NAR) is an emerging field that seeks to design neural networks that mimic classical algorithmic computations. Today, graph neural networks (GNNs) are widely used in neural algorithmic reasoners due to their message passing framework and permutation equivariance. In this extended abstract, we challenge this design choice, and replace the equivariant aggregation function with a *recurrent neural network*. While seemingly counter-intuitive, this approach has appropriate grounding when nodes have a natural ordering—and this is the case frequently in established reasoning benchmarks like CLRS-30. Indeed, our recurrent NAR (RNAR) model performs very strongly on such tasks, while handling many others gracefully. A notable achievement of RNAR is its decisive state-of-the-art result on the Heapsort and Quickselect tasks, both deemed as a significant challenge for contemporary neural algorithmic reasoners—especially the latter, where RNAR achieves a mean micro-F_1 score of 87%.

1 Introduction

Neural algorithmic reasoning [1, NAR] is an area of research that explores how *neural networks* can learn *algorithms* from data. This seeks to combine the benefits of both neural networks and classical algorithms and gives rise to the possibility of designing better neural networks that can learn and develop stronger algorithms for challenging real-world reasoning problems [2–7].

Graph neural networks [8, GNNs] are the most commonly used class of models in NAR due to their *algorithmic alignment* [9] to dynamic programming [10]. Algorithmic alignment is the observation that an increase in the structural similarity between an algorithm and a neural network tends to result in an increase in the neural network's ability to learn the algorithm—and GNNs can offer a high degree of flexibility in how this alignment is designed [11, 12]. Indeed, GNNs are capable of generalising out-of-distribution (OOD) on standard algorithmic benchmarks like CLRS [13] to a significantly higher degree [14] than, e.g., Transformer-based LLMs [15, 16].

While it is evident that this improvement is largely due to the *permutation equivariance* properties of GNNs [17], so much so that often it is important for OOD generalisation to leverage strictly less expressive categories of permutation equivariant aggregators [11, 18], it is also worth noting that such an approach forces all neighbours of a node to be treated *symmetrically*—and many tasks of interest to algorithms do not include such a symmetry. This is especially the case for *sequential* algorithms, where the input comes in the form of a *list* and hence a natural ordering between the elements exists. Indeed, such algorithms are frequent in CLRS—ten out of thirty of its tasks [19–23] are sequential.

In this extended abstract, we detail our attempt to leverage a *recurrent* aggregator in a state-of-the-art neural algorithmic reasoning architecture (leaving all other components the same). Specifically, we leverage long short-term memory (LSTM) networks [24, 25] as the aggregation function. The resulting *recurrent NAR* (**RNAR**) model yielded a *serendipitous* discovery: it significantly outperformed prior art on many sequential tasks in CLRS, while also gracefully handling many algorithms without such a bias! Further, RNAR sets a dominating state-of-the-art result on the Quickselect task [21], which was previously identified as a major open challenge in neural algorithmic reasoning [26].

Kaijia Xu and Petar Veličković, Recurrent Aggregators in Neural Algorithmic Reasoning (Extended Abstract). Presented at the Third Learning on Graphs Conference (LoG 2024), Virtual Event, November 26–29, 2024.

2 Towards RNAR

Let $G = (V, E)$ denote a graph, where V is the set of vertices and E is the set of edges in the graph. Let the one-hop neighbourhoods of node u be defined as $\mathcal{N}_u = \{v \in V \mid (v, u) \in E\}$, and $\mathbf{x}_u \in \mathbb{R}^k$ be the features of node u. With reference to the definition of GNNs in Bronstein et al. [17], we can formalise the message passing framework over this graph as:

$$\mathbf{x}'_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right). \tag{1}$$

The message function $\psi : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^m$ first computes the message to be sent along edge (v, u) based on node u and its neighbour node v. Then, the receiver node u will aggregate the messages along incoming edges using the aggregation function $\bigoplus : \text{bag}(\mathbb{R}^m) \to \mathbb{R}^m$. Lastly, the update function $\phi : \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^k$ updates the features of the receiver node u based on its current features and the aggregated messages. Typically, ψ and ϕ are deep multilayer perceptrons.

While various forms of $\mathcal{N}_u$ have been explored by prior art [27], nowadays it is standard practice to use a *fully-connected graph* [13], i.e. $\mathcal{N}_u = V$, and allow the GNN to infer the important neighbours by itself. This assumption also makes it easier to learn multiple algorithms in the same model [14].

2.1 Aggregation functions

The choice of aggregation function, $\bigoplus$, is often central to NARs' success. While it is well-known that aggregators such as summing are provably powerful for structural tasks [28], in practice a more aligned choice—such as $\max$—tends to be superior, especially out-of-distribution [11, 18]. In nearly all cases, $\bigoplus$ is chosen to be *permutation invariant*—i.e. yielding identical answers for any permutation of neighbours. Such models are known to be universal under certain conditions [29, 30].

Permutation invariance is challenging to learn from data due to the high degrees-of-freedom induced by the permutation group and, as such, it is believed that this is a key reason for why GNNs tend to extrapolate better on algorithmic tasks compared to autoregressive Transformers [15]. Invariance to permutations also grants the model invariance to a certain kind of asynchronous execution [12].

2.2 Why would we ever *drop* permutation invariance in NARs?

With all of the above reasons in favour, it might seem extremely counter-intuitive to ever consider setting $\bigoplus$ to something which is not permutation invariant. So, *why did we even bother attempting it*?

There are three key reasons:

- Firstly, permutation invariance is a property typically most desired when inputs are assumed to be given *without any order*. In many algorithmic tasks, this is frequently enough not the case, making this direction worth studying. Many classical algorithm categories, such as *sorting* [23] and *searching* [21], assume that an input is a *list*, inducing a natural order[1] between the nodes. Previous research [31] highlighted how such *sequential* algorithms are not favourable for GNNs.
- Secondly, imposing permutation symmetry forces *all* neighbours to be treated *equivalently*, limiting expressive power and the scope of functions that could be learnt. Recently there have been trends to eliminate various kinds of equivariances from models, leading to surprising improvements [32, 33], which may also be considered motivating for our attempt.
- Lastly, using a permutation-invariant aggregator is typically realised by fixing a *commutative monoid* structure. If the target task requires a substantially different monoid choice—often the case in more complex tasks—this can pose a unique challenge for NARs [34].

2.3 The RNAR architecture

Motivated by these reasons, in RNAR we drop the commutative monoid assumption, and instead treat $\bigoplus : \text{list}(\mathbb{R}^m) \to \mathbb{R}^m$ as an arbitrary *list reduction* function. We will hence assume that the $N = |V|$ node features are pre-arranged in a list $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$. Such an ordering will always be provided by the CLRS benchmark through its `pos` node input feature [13].

[1] We will study how important is to exploit this natural order in Appendix B.

A popular, theoretically expressive choice of such a sequential aggregator is the long short-term memory (LSTM) network [24], which we employ in this work.

In a typical fully-connected GNN, each node receives messages from all other nodes, and therefore receives a total of N messages in one computational step. In a GNN with an LSTM as its aggregation function, the messages from the neighbouring nodes are fed into the LSTM in a particular order.

The LSTM will therefore run for N time steps, with the input to the LSTM at each time step, $1 \leq t \leq N$, being one of the N messages computed using ψ:

$$\mathbf{z}_t^{(u)} = \text{LSTM}\left(\psi(\mathbf{x}_u, \mathbf{x}_t), \mathbf{z}_{t-1}^{(u)}\right) \tag{2}$$

where the initial LSTM cell state, $\mathbf{z}_0^{(u)}$, is initialised to a fixed zero vector, $\mathbf{0}$. The final updated embeddings are then computed using the output of the LSTM at the last time step, which is considered to be the aggregation of the N messages:

$$\mathbf{x}'_u = \phi\left(\mathbf{x}_u, \mathbf{z}_N^{(u)}\right) \tag{3}$$

Since the choice of the initial node ordering clearly affects $\mathbf{z}_N^{(u)}$, LSTM as an aggregator is not invariant to message receiving order, breaking permutation invariance.

While our work offers the first comprehensive study of such a recurrent aggregator on a benchmark like CLRS—and reveals surprising results—we stress that we are far from the first work to attempt replacing a GNN's aggregator with a recurrent neural network.

Key works to consider here include GraphSAGE [35]—one of the earliest GNNs to attempt an LSTM aggregator; Janossy pooling [36] and PermGNN [37]—illustrating how such models can be made permutation equivariant in expectation by applying them to several sampled permutations; and LCM [34]—which showed GRU aggregators [38] can effectively learn a challenging commutative monoid. Note that RNAR uses a stronger base model than GraphSAGE; we ablate against this in Appendix A.

3 Evaluating RNAR

We evaluate RNAR using the CLRS-30 algorithmic reasoning benchmark [13]. Since we want to examine whether the use of RNAR enables the emergence of novel capabilities not covered by previous state-of-the-art, we insert RNAR into the state-of-the-art Triplet-GMPNN architecture [14] with hint reversals [39], and compare its performance against the baseline Triplet-GMPNN, as well as two additional state-of-the-art neural algorithmic executors, which both offer inventive ways to boost performance: Relational Transformers [40] and G-ForgetNets [41].

We remark that there are several very interesting recent works improving NARs [42, 43] which we exclude because they leverage a different learning regime and/or CLRS environment assumptions.

We commence with Table 1, showcasing RNAR's performance on *sequential* algorithms in CLRS-30, where it is expected it could perform favourably in spite of its lack of symmetry.

Table 1: Micro-F_1 test OOD scores on sequential algorithms. RNAR improves on its Triplet-GMPNN baseline on 8/10 of them (underlined) and sets new state-of-the-art on 6/10.

Algorithm	Triplet-GMPNN	RT	G-ForgetNet	RNAR
Activity Selector	$95.18\% \pm 0.45$	$87.72\% \pm 2.7$	**99.03**$\% \pm 0.10$	$\underline{95.23}\% \pm 0.71$
Binary Search	$77.58\% \pm 2.35$	$81.48\% \pm 6.7$	**85.96**$\% \pm 1.59$	$64.71\% \pm 6.79$
Bubble Sort	$80.51\% \pm 9.10$	$38.22\% \pm 13.0$	$83.19\% \pm 2.59$	$\underline{\textbf{95.78}}\% \pm 0.40$
Find Max. Subarray	$76.36\% \pm 0.43$	$66.52\% \pm 3.7$	$78.97\% \pm 0.70$	$\underline{\textbf{83.53}}\% \pm 2.17$
Heapsort	$49.13\% \pm 10.35$	$32.96\% \pm 14.8$	$57.47\% \pm 6.08$	$\underline{\textbf{93.07}}\% \pm 1.03$
Insertion Sort	$87.21\% \pm 2.80$	$89.43\% \pm 9.0$	**98.40**$\% \pm 0.21$	$\underline{93.00}\% \pm 1.77$
Minimum	$98.43\% \pm 0.01$	$95.28\% \pm 2.0$	**99.26**$\% \pm 0.08$	$96.92\% \pm 0.09$
Quickselect	$0.47\% \pm 0.25$	$19.18\% \pm 17.3$	$6.30\% \pm 0.85$	$\underline{\textbf{87.08}}\% \pm 2.21$
Quicksort	$85.69\% \pm 4.53$	$39.42\% \pm 13.2$	$73.28\% \pm 6.25$	$\underline{\textbf{94.73}}\% \pm 0.63$
Task Scheduling	$87.25\% \pm 0.35$	$82.93\% \pm 1.8$	$84.55\% \pm 0.35$	$\underline{\textbf{88.08}}\% \pm 1.30$

The most important result is clearly on the *Quickselect* task, wherein RNAR **sets the best recorded micro-F_1 score by a wide margin**, settling an important open challenge [26]. Besides this, RNAR is highly potent for sorting algorithms, and generally outperforms its Triplet-GMPNN baseline in nearly all of the sequential tasks.

Armed with this exciting result, we now evaluate RNAR on *all* other tasks in CLRS-30 – see Table 2.

Table 2: Single-task OOD average micro-F_1 score of previous SOTA: Triplet-GMPNN, RT and G-ForgetNet and our new RNAR model. For four of the algorithms (marked in red), RNAR with triplets ran out of memory on a V100 GPU, and an MPNN model [44] was used as a basis instead.

Algorithm	Triplet-GMPNN	RT	G-ForgetNet	RNAR
Activity Selector	95.18% ± 0.45	87.72% ± 2.7	**99.03**% ± 0.10	95.23% ± 0.71
Articulation Points	91.04% ± 0.92	34.15% ± 14.6	**97.97**% ± 0.46	26.32% ± 27.34
Bellman-Ford	97.39% ± 0.19	94.24% ± 1.5	**99.18**% ± 0.11	96.00% ± 0.38
BFS	99.93% ± 0.03	99.14% ± 0.7	99.96% ± 0.01	**100.0**% ± 0.0
Binary Search	77.58% ± 2.35	81.48% ± 6.7	**85.96**% ± 1.59	64.71% ± 6.79
Bridges	97.70% ± 0.34	37.88% ± 11.8	**99.43**% ± 0.15	72.22% ± 12.66
Bubble Sort	80.51% ± 9.10	38.22% ± 13.0	83.19% ± 2.59	**95.78**% ± 0.40
DAG Shortest Paths	98.19% ± 0.30	96.61% ± 1.6	**99.37**% ± 0.03	96.40% ± 1.47
DFS	**100.0**% ± 0.00	39.23% ± 10.5	74.31% ± 5.03	**100.0**% ± 0.00
Dijkstra	96.05% ± 0.60	91.20% ± 5.8	**99.14**% ± 0.06	95.04% ± 1.62
Find Max. Subarray	76.36% ± 0.43	66.52% ± 3.7	78.97% ± 0.70	**83.53**% ± 2.17
Floyd-Warshall	48.52% ± 1.04	31.59% ± 7.6	**56.32**% ± 0.86	27.49% ± 6.95
Graham Scan	93.62% ± 0.91	74.15% ± 7.4	**97.67**% ± 0.14	76.20% ± 4.51
Heapsort	49.13% ± 10.35	32.96% ± 14.8	57.47% ± 6.08	**93.07**% ± 1.03
Insertion Sort	87.21% ± 2.80	89.43% ± 9.0	**98.40**% ± 0.21	93.00% ± 1.77
Jarvis' March	91.01% ± 1.30	**94.57**% ± 2.2	88.53% ± 2.96	91.83% ± 1.77
Knuth-Morris-Pratt	**19.51**% ± 4.57	0.03% ± 0.1	12.45% ± 3.12	4.54% ± 2.60
LCS Length	80.51% ± 1.84	83.32% ± 4.1	**85.43**% ± 0.47	66.91% ± 2.53
Matrix Chain Order	91.68% ± 0.59	**91.89**% ± 1.2	91.08% ± 0.51	25.12% ± 1.86
Minimum	98.43% ± 0.01	95.28% ± 2.0	**99.26**% ± 0.08	96.92% ± 0.09
MST-Kruskal	89.93% ± 0.43	64.91% ± 11.8	**91.25**% ± 0.40	67.29% ± 0.93
MST-Prim	87.64% ± 1.79	85.77% ± 7.9	**95.19**% ± 0.33	86.60% ± 4.42
Naïve String Matcher	78.67% ± 4.99	65.01% ± 32.3	**97.02**% ± 0.77	93.71% ± 2.26
Optimal BST	73.77% ± 1.48	74.40% ± 2.6	**83.58**% ± 0.49	36.04% ± 12.55
Quickselect	0.47% ± 0.25	19.18% ± 17.3	6.30% ± 0.85	**87.08**% ± 2.21
Quicksort	85.69% ± 4.53	39.42% ± 13.2	73.28% ± 6.25	**94.73**% ± 0.63
Segments Intersect	97.64% ± 0.09	84.94% ± 2.6	**99.06**% ± 0.39	97.30% ± 0.29
SCC	43.43% ± 3.15	28.59% ± 15.2	**53.53**% ± 2.48	48.43% ± 8.01
Task Scheduling	87.25% ± 0.35	82.93% ± 1.8	84.55% ± 0.35	**88.08**% ± 1.30
Topological Sort	87.27% ± 2.67	80.62% ± 17.5	**99.92**% ± 0.02	74.00% ± 8.18
Overall average	80.04%	66.18%	**82.89**%	75.78%

While it is evident that removing permutation invariance does *not* yield the strongest model overall, we found that performance regressions compared to Triplet-GMPNNs were not as common as expected, and only 4% average performance points were lost compared to them.

Still, RNAR proves itself a worthy element in the NAR toolbox: with its outperformances on Find Max Subarray, Heapsort and especially Quickselect, there are now only *three* tasks in CLRS-30 (Floyd-Warshall, Knuth-Morris-Pratt and Strongly Connected Components) for which there is no known OOD result above 80%—indicating that we soon may need a new test split for CLRS-30.

As such, it is our hope that RNAR inspires future research into non-commutative aggregators in NAR. We note two obvious limitations worth exploring in the future: the memory considerations of LSTM aggregators, which caused OOMs in conjunction with triplets on four of the tasks (see also Appendix C), and the fact that the Knuth-Morris-Pratt algorithm proves challenging in spite of being a string algorithm. For the former, one may consider alternatives to recurrent aggregators such as Binary-GRUs [34]; for the latter, seeking out better alignment with *automata* may be desirable.

References

[1] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7), 2021. 1

[2] Andreea-Ioana Deac, Petar Veličković, Ognjen Milinkovic, Pierre-Luc Bacon, Jian Tang, and Mladen Nikolic. Neural algorithmic reasoners are implicit planners. *Advances in Neural Information Processing Systems*, 34:15529–15542, 2021. 1

[3] Petar Veličković, Matko Bošnjak, Thomas Kipf, Alexander Lerchner, Raia Hadsell, Razvan Pascanu, and Charles Blundell. Reasoning-modulated representations. In *Learning on Graphs Conference*, pages 50–1. PMLR, 2022.

[4] Yu He, Petar Veličković, Pietro Liò, and Andreea Deac. Continuous neural algorithmic planners. In *Learning on Graphs Conference*, pages 54–1. PMLR, 2022.

[5] Danilo Numeroso, Davide Bacciu, and Petar Veličković. Dual algorithmic reasoning. *arXiv preprint arXiv:2302.04496*, 2023.

[6] Dobrik Georgiev, Ramon Vinas, Sam Considine, Bianca Dumitrascu, and Pietro Lio. Narti: Neural algorithmic reasoning for trajectory inference. In *The 2023 ICML Workshop on Computational Biology*, 2023.

[7] Dobrik Georgiev Georgiev, Danilo Numeroso, Davide Bacciu, and Pietro Liò. Neural algorithmic reasoning for combinatorial optimisation. In *Learning on Graphs Conference*, pages 28–1. PMLR, 2024. 1

[8] Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, 2023. 1

[9] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019. 1

[10] Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. *Advances in neural information processing systems*, 35:20635–20647, 2022. 1, 7

[11] Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. How neural networks extrapolate: From feedforward to graph neural networks. *arXiv preprint arXiv:2009.11848*, 2020. 1, 2

[12] Andrew Joseph Dudzik, Tamara von Glehn, Razvan Pascanu, and Petar Veličković. Asynchronous algorithmic alignment with cocycles. In *Learning on Graphs Conference*, pages 3–1. PMLR, 2024. 1, 2

[13] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pages 22084–22102. PMLR, 2022. 1, 2, 3, 7

[14] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on graphs conference*, pages 2–1. PMLR, 2022. 1, 2, 3, 7

[15] Larisa Markeeva, Sean McLeish, Borja Ibarz, Wilfried Bounsi, Olga Kozlova, Alex Vitvitskyi, Charles Blundell, Tom Goldstein, Avi Schwarzschild, and Petar Veličković. The clrs-text algorithmic reasoning language benchmark. *arXiv preprint arXiv:2406.04229*, 2024. 1, 2

[16] Wilfried Bounsi, Borja Ibarz, Andrew Dudzik, Jessica B Hamrick, Larisa Markeeva, Alex Vitvitskyi, Razvan Pascanu, and Petar Veličković. Transformers meet neural algorithmic reasoners. *arXiv preprint arXiv:2406.09308*, 2024. 1

[17] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021. 1, 2

[18] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*, 2019. 1, 2

[19] Jon Bentley. Programming pearls: algorithm design techniques. *Communications of the ACM*, 27(9):865–873, 1984. 1

[20] Fănică Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal on Computing*, 1(2): 180–187, 1972.

[21] Charles AR Hoare. Algorithm 65: find. *Communications of the ACM*, 4(7):321–322, 1961. 1, 2

[22] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[23] John William Joseph Williams. Algorithm 232: heapsort. *Commun. ACM*, 7:347–348, 1964. 1, 2

[24] Jürgen Schmidhuber, Sepp Hochreiter, et al. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997. 1, 3

[25] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000. 1

[26] Michael Galkin. Graph & Geometric ML in 2024: Where We Are and What's Next (Part II — Applications), 2024. URL https://towardsdatascience.com/graph-geometric-ml-in-2024-where-we-are-and-whats-next-part-ii-applications-1ed786f7bf63. 1, 4

[27] Petar Veličković, Lars Buesing, Matthew Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. *Advances in Neural Information Processing Systems*, 33: 2232–2244, 2020. 2

[28] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 2

[29] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017. 2

[30] Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Michael A Osborne, and Ingmar Posner. Universal approximation of functions on sets. *Journal of Machine Learning Research*, 23(151): 1–56, 2022. 2

[31] Valerie Engelmayer, Dobrik Georgiev Georgiev, and Petar Veličković. Parallel algorithms align with neural execution. In *Learning on Graphs Conference*, pages 31–1. PMLR, 2024. 2

[32] Yuyang Wang, Ahmed AA Elhag, Navdeep Jaitly, Joshua M Susskind, and Miguel Ángel Bautista. Swallowing the bitter pill: Simplified scalable conformer generation. In *Forty-first International Conference on Machine Learning*, 2024. 2

[33] Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. *Nature*, pages 1–3, 2024. 2

[34] Euan Ong and Petar Veličković. Learnable commutative monoids for graph neural networks. In *Learning on Graphs Conference*, pages 43–1. PMLR, 2022. 2, 3, 4

[35] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 3, 7

[36] Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018. 3, 8

[37] Indradyumna Roy, Abir De, and Soumen Chakrabarti. Adversarial permutation guided node representations for link prediction. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 9445–9453, 2021. 3

[38] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014. 3

[39] Beatrice Bevilacqua, Kyriacos Nikiforou, Borja Ibarz, Ioana Bica, Michela Paganini, Charles Blundell, Jovana Mitrovic, and Petar Veličković. Neural algorithmic reasoning with causal regularisation. In *International Conference on Machine Learning*, pages 2272–2288. PMLR, 2023. 3

[40] Cameron Diao and Ricky Loynd. Relational attention: Generalizing transformers for graph-structured tasks. *arXiv preprint arXiv:2210.05062*, 2022. 3

[41] Montgomery Bohde, Meng Liu, Alexandra Saxton, and Shuiwang Ji. On the markov property of neural algorithmic reasoning: Analyses and methods. *arXiv preprint arXiv:2403.04929*, 2024. 3

[42] Sadegh Mahdavi, Kevin Swersky, Thomas Kipf, Milad Hashemi, Christos Thrampoulidis, and Renjie Liao. Towards better out-of-distribution generalization of neural algorithmic reasoning tasks. *arXiv preprint arXiv:2211.00692*, 2022. 3

[43] Gleb Rodionov and Liudmila Prokhorenkova. Discrete neural algorithmic reasoning. *arXiv preprint arXiv:2402.11628*, 2024. 3

[44] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 4, 7

[45] Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. The neural data router: Adaptive control flow in transformers improves systematic generalization. *arXiv preprint arXiv:2110.07732*, 2021. 7

[46] Adi Haviv, Ori Ram, Ofir Press, Peter Izsak, and Omer Levy. Transformer language models without positional encodings still learn positional information. *arXiv preprint arXiv:2203.16634*, 2022. 7

A RNAR ablations on base model and positional information

In this Appendix, we aim to answer two key questions about the choice of RNAR base model, as well as its reliance on positional information—both enumerated in Table 3. The results are taken across only the algorithms where RNAR does not run out of memory, in order to ensure a meaningful ablation of capabilities.

The ablations are as follows:

RNAR gains additional power through architectural choices. As already previously discussed, RNAR is not the first GNN architecture to use a recurrent aggregator—GraphSAGE [35] being a notable early example. However, we remark that RNAR is not the same architecture as GraphSAGE—its base model relies on architectural novelties such as triplet messages [10] and gating mechanisms [45], as described by Ibarz et al. [14].

We argue that these improvements further amplify the impact of a recurrent aggregator, and compare against RNAR-MPNN, a model with a recurrent aggregator which otherwise uses a more typical message-passing neural network [13, 44] as the base model. Such an architecture is similar to GraphSAGE-LSTM.

As is evident in Table 3, introducing the architectural changes results in an 11% increase in average OOD execution performance, which is evidence in support of our hypothesis. That being said, the simpler MPNN architecture does appear to better align with certain classes of algorithms, such as dynamic programming (see LCS Length, Matrix Chain Order and Optimal BST) as well as the Floyd-Warshall and Knuth-Morris-Pratt algorithms; these discrepancies may warrant further investigation into the optimal base model for RNAR.

RNAR can meaningfully exploit positional features. Since nodes are fed into the LSTM aggregator of RNAR in a canonical order (using its positional feature), it may be argued that the model does not need to use this feature anymore. We believe that positional information may still be quite relevant, especially in graph algorithms where nontrivial tiebreaking is common. To assess this, we compare RNAR against a variant which does not use the positional feature (akin to NoPE [46]).

Once again, the results in Table 3 provide evidence for our claim, with RNAR losing 5% average performance when the positional features are withheld. That being said, this removal does seem to provide meaningful uplift on nearly all *sorting algorithms*, which may provide useful motivation for future investigation into how the positional feature may be (mis)used by models like RNAR.

Table 3: Test results of RNAR, RNAR-MPNN (an MPNN-based architecture with a recurrent aggregator), and RNAR-NoPE (RNAR without positional inputs) on all algorithms where RNAR does not run out of memory.

Algorithm	RNAR	RNAR-MPNN	RNAR-NoPE
Activity Selector	$95.23\% \pm 0.71$	$87.35\% \pm 4.19$	**96.86**$\% \pm 0.31$
Bellman-Ford	**96.00**$\% \pm 0.38$	$93.54\% \pm 0.92$	$95.33\% \pm 0.10$
BFS	**100.00**$\% \pm 0.00$	**100.00**$\% \pm 0.00$	$89.32\% \pm 3.96$
Binary Search	**64.71**$\% \pm 6.79$	$52.75\% \pm 4.17$	$62.40\% \pm 9.14$
Bubble Sort	$95.78\% \pm 0.40$	$62.79\% \pm 9.68$	**96.86**$\% \pm 0.35$
DAG Shortest Paths	**96.40**$\% \pm 1.47$	$49.06\% \pm 4.98$	$83.07\% \pm 6.41$
DFS	**100.0**$\% \pm 0.00$	$6.78\% \pm 2.30$	$97.37\% \pm 2.15$
Dijkstra	$95.04\% \pm 1.62$	**96.75**$\% \pm 0.49$	$86.38\% \pm 6.67$
Find Max. Subarray	**83.53**$\% \pm 2.17$	$73.47\% \pm 0.79$	$75.43\% \pm 5.67$
Floyd-Warshall	$27.49\% \pm 6.95$	**41.33**$\% \pm 4.56$	$9.92\% \pm 4.50$
Graham Scan	$76.20\% \pm 4.51$	$50.92\% \pm 3.04$	**79.18**$\% \pm 2.25$
Heapsort	$93.07\% \pm 1.03$	$67.61\% \pm 10.71$	**95.41**$\% \pm 0.33$
Insertion Sort	$93.00\% \pm 1.77$	$87.88\% \pm 0.75$	**99.16**$\% \pm 0.27$
Knuth-Morris-Pratt	$4.54\% \pm 2.60$	**23.01**$\% \pm 10.60$	$3.96\% \pm 1.62$
LCS Length	$66.91\% \pm 2.53$	**77.46**$\% \pm 2.83$	$73.29\% \pm 4.12$
Matrix Chain Order	$25.12\% \pm 1.86$	**43.55**$\% \pm 9.14$	$24.18\% \pm 1.92$
Minimum	$96.92\% \pm 0.09$	**98.08**$\% \pm 1.16$	$97.73\% \pm 0.38$
MST-Prim	$86.60\% \pm 4.42$	$87.59\% \pm 3.52$	**89.82**$\% \pm 1.97$
Naïve String Matcher	**98.95**$\% \pm 0.42$	$28.55\% \pm 21.13$	$18.81\% \pm 9.13$
Optimal BST	$36.04\% \pm 12.55$	**42.29**$\% \pm 13.39$	$22.31\% \pm 13.06$
Quickselect	**87.08**$\% \pm 2.21$	$83.90\% \pm 3.11$	$79.67\% \pm 5.54$
Quicksort	**94.73**$\% \pm 0.63$	$71.57\% \pm 3.34$	$94.66\% \pm 0.43$
Segments Intersect	$97.30\% \pm 0.29$	**97.84**$\% \pm 0.15$	$97.03\% \pm 0.21$
SCC	**48.43**$\% \pm 8.01$	$28.53\% \pm 3.01$	$45.35\% \pm 11.01$
Task Scheduling	**88.08**$\% \pm 1.30$	$81.65\% \pm 0.59$	$87.89\% \pm 1.34$
Topological Sort	$74.00\% \pm 8.18$	**81.98**$\% \pm 14.07$	$76.29\% \pm 9.01$
Overall average	**77.74**%	66.01%	72.22%

B RNAR's robustness against choice of node permutation

One of the motivating factors for the suitability of recurrent aggregators in NAR is the fact that nodes may often have a *canonical ordering* when executing algorithms—and this is certainly the case in CLRS-30. We now seek to investigate how relevant is this canonicalisation to the model's performance, by checking how well it performs when node permutations are consistently *randomly* sampled, during both training and inference.

It might be noted that aggregating across randomly sampled permutations is exactly one of the strategies employed by Janossy pooling [36] to achieive permutation equivariance in expectation. This motivates our comparison in Table 4, where we benchmark RNAR (using canonical node order) against RNAR-Janossy-k, which chooses k random permutations, runs the LSTM over each of them, and averages the resulting embedding vectors. We focus on values of $k \in \{1, 2, 3\}$, to provide a meaningful indication of trends without requiring too many computational resources.

As in Appendix A, we only take results across the algorithms where RNAR-Janossy-3 does not run out of memory, in order to ensure a meaningful ablation.

While our results in Table 4 do indicate a slight average advantage to the canonical order used by RNAR, the comparison is substantially less clear-cut; there are several algorithms from which there is a very clear uplift from regularising RNAR towards permutation equivariance in this way. If we take into consideration the number of algorithms where Janossy pooling runs out of memory, it may still be concluded that canonicalisation is the better choice. That being said, our ablation points at a clear line of future work, which would study principled ways to regularise NARs without symmetries (such as permutation equivariance) towards satisfying these symmetries in expectation.

Table 4: Test results of RNAR and RNAR-Janossy-k (where embeddings are aggregated across k random node permutations) models on all algorithms where these ablations do not run out of memory.

Algorithm	**RNAR**	**RNAR-Janossy-1**	**RNAR-Janossy-2**	**RNAR-Janossy-3**
Activity Selector	$95.23\% \pm 0.71$	$95.54\% \pm 0.93$	$95.65\% \pm 0.64$	**96.13**$\% \pm 0.99$
Bellman-Ford	$96.00\% \pm 0.38$	$96.31\% \pm 0.43$	**96.64**$\% \pm 0.05$	$96.22\% \pm 0.40$
BFS	**100.00**$\% \pm 0.00$	**100.00**$\% \pm 0.00$	**100.00**$\% \pm 0.00$	**100.00**$\% \pm 0.00$
Binary Search	**64.71**$\% \pm 6.79$	$55.40\% \pm 6.96$	$36.37\% \pm 4.02$	$60.60\% \pm 7.16$
DAG Shortest Paths	**96.40**$\% \pm 1.47$	$89.71\% \pm 4.17$	$80.90\% \pm 6.16$	$84.40\% \pm 5.76$
DFS	**100.0**$\% \pm 0.00$	$17.52\% \pm 4.11$	$18.98\% \pm 2.25$	$39.53\% \pm 9.44$
Dijkstra	$95.04\% \pm 1.62$	$89.67\% \pm 5.44$	$92.33\% \pm 1.24$	**96.54**$\% \pm 1.09$
Find Max. Subarray	**83.53**$\% \pm 2.17$	$67.93\% \pm 7.39$	$78.64\% \pm 1.02$	$79.14\% \pm 2.28$
Floyd-Warshall	$27.49\% \pm 6.95$	**70.36**$\% \pm 10.24$	$46.70\% \pm 18.75$	$47.86\% \pm 17.72$
Graham Scan	$76.20\% \pm 4.51$	$83.59\% \pm 4.75$	$86.61\% \pm 3.22$	**90.88**$\% \pm 0.78$
Insertion Sort	$93.00\% \pm 1.77$	$94.72\% \pm 0.56$	**97.44**$\% \pm 0.86$	$95.01\% \pm 1.39$
Knuth-Morris-Pratt	$4.54\% \pm 2.60$	$0.67\% \pm 0.28$	**8.98**$\% \pm 2.54$	$3.65\% \pm 1.08$
LCS Length	$66.91\% \pm 2.53$	$82.97\% \pm 2.00$	**84.78**$\% \pm 0.07$	$78.84\% \pm 2.98$
Matrix Chain Order	$25.12\% \pm 1.86$	$84.94\% \pm 2.79$	**86.59**$\% \pm 3.35$	$82.97\% \pm 4.30$
Minimum	$96.92\% \pm 0.09$	**97.20**$\% \pm 0.29$	$93.81\% \pm 1.76$	$85.63\% \pm 9.86$
MST-Prim	$86.60\% \pm 4.42$	$91.59\% \pm 0.96$	**91.72**$\% \pm 1.12$	$90.93\% \pm 2.81$
Naïve String Matcher	**98.95**$\% \pm 0.42$	$5.63\% \pm 2.30$	$61.40\% \pm 24.26$	$11.75\% \pm 5.74$
Optimal BST	$36.04\% \pm 12.55$	$50.21\% \pm 20.45$	**78.06**$\% \pm 3.40$	$70.51\% \pm 9.91$
Segments Intersect	**97.30**$\% \pm 0.29$	$92.17\% \pm 1.57$	$91.45\% \pm 2.14$	$94.22\% \pm 1.60$
Task Scheduling	$88.08\% \pm 1.30$	$88.18\% \pm 0.53$	**88.55**$\% \pm 0.91$	$87.74\% \pm 0.96$
Topological Sort	$74.00\% \pm 8.18$	**95.13**$\% \pm 1.11$	$73.08\% \pm 12.39$	$74.55\% \pm 8.79$
Overall average	**76.29**%	73.78%	75.65%	74.63%

C Timing performance of RNAR

In Figure 1 we show the effects of adding RNAR on computation time compared to the baseline Triplet-GMPNN. As expected, the overall training steps-per-second is worse affected for algorithms that require more intermediate iterations before arriving at the final answer.

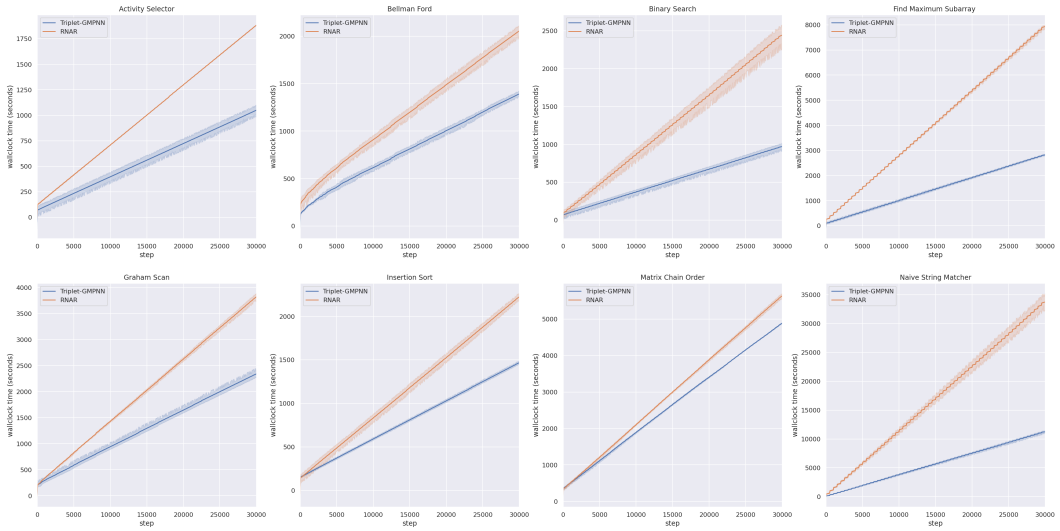

Figure 1: A comparison of the wall-clock times (y-axis, in seconds) required for completing a certain number of training steps (x-axis), for the baseline Triplet-GMPNN (in blue) against RNAR (in orange), across eight representative algorithms in CLRS-30.