Fixing Edge-Pointer Compositionality in Neural Algorithmic Reasoning

Stjepan Požgaj [1] **Dobrik Georgiev** [2] **Marin Šilić** [1] **Goran Delač** [1] **Klemo Vladimir** [1]

Abstract

Neural algorithmic reasoning (NAR) aims to train neural networks that emulate classical algorithms. Despite substantial progress on most of CLRS-30, Floyd–Warshall remains one of the hardest tasks. This is surprising given that standard NAR architectures already emphasize triplet reasoning aligned with that algorithm. In this work, we identify an information bottleneck in how the standard CLRS-30 encoder and decoder represent edge-pointer probes. We remove this bottleneck by introducing higher-order information flow, improving the performance of the baseline NAR model on Floyd–Warshall from 46% to 95% and on Matrix Chain Order from 91% to 98%. Our results suggest that proper compositionality matters: the hardest failures in NAR can lie in interfaces and encodings, not only in the capacity or inductive bias of the processor itself.

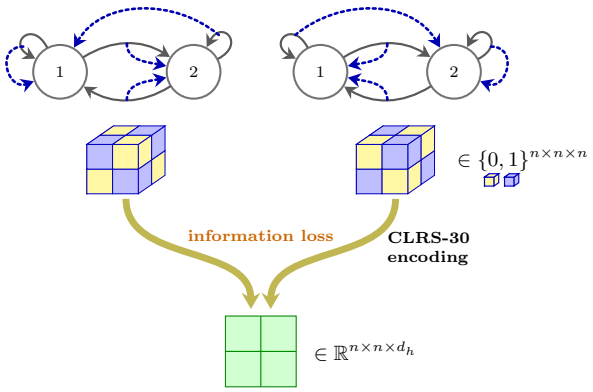

Figure 1. Two edge-pointer assignments are shown both on a two-node graph ($n = 2$) and as binary tensors $X^{(a)}, X^{(b)} \in \{0,1\}^{n\times n\times n}$. Dashed arrows indicate to which node each directed edge points. Although the assignments are different, the default CLRS-30 edge-pointer encoder maps both to the same edge representation $\mathbf{u}_X \in \mathbb{R}^{n\times n\times d_h}$. This motivates us to redefine the edge-pointer encoding-decoding scheme.

1. Introduction

Neural algorithmic reasoning (NAR) (Veličković et al., 2022) trains neural networks to emulate classical algorithms, with the ambition of obtaining systems that generalise well beyond their training distribution and beyond the abstract representations of algorithms. The standard paradigm is *encode-process-decode* (Veličković et al., 2022; Ibarz et al., 2022): lightweight encoders map algorithmic probes into a shared latent space, a *processor* – typically a graph neural network (GNN) – performs iterative computation, and decoders translate the resulting latents back into discrete algorithmic predictions. A key desideratum of this framework is *algorithmic alignment* (Dudzik & Veličković, 2022; Xu et al., 2021): the processor's message-passing structure should mirror the computational structure of the target algorithm, yielding better sample efficiency and out-of-distribution (OOD) generalisation.

[1]University of Zagreb, Faculty of Electrical Engineering and Computing [2]Graphcore Research. Correspondence to: Stjepan Požgaj <stjepan.pozgaj@fer.unizg.hr>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. Copyright 2026 by the author(s).

A longer-term motivation is for the processor to serve as the algorithmic core of *perception-to-reasoning* pipelines: once pre-trained on abstract algorithmic trajectories, it can be coupled with task-specific encoders and decoders that translate raw, real-world inputs into the processor's latent space and back (He et al., 2026).

Despite the rapid progress surveyed in Section 4, the *encoder* and *decoder* of the NAR pipeline have received comparatively little scrutiny. The canonical design from Veličković et al. (2022) encodes probe tensors in isolation, and subsequently computes node/edge/graph features, aggregating the probe tensors depending on the probe location and datatype (see Section 2.1). For most probe types this causes no information loss. For *edge-pointer probes* – central to Floyd–Warshall, Matrix Chain Order, and Optimal BST – our results support a *compositional-learning* view: progress can come not only from improving processor architectures or training strategies, but also from improving how the encode-process-decode modules compose. Concretely, our first contribution is **proving that the standard edge-pointer encoder is non-injective**: distinct pointer states can collapse to identical edge embeddings (Section 2.4),

a structural flaw that has persisted across the community, affecting even methods that have otherwise withstood the test of time (Ibarz et al., 2022; Veličković et al., 2020).

We identify a second, related bottleneck at the processor–decoder interface. The Triplet-GMPNN processor (Ibarz et al., 2022), the most widely used NAR architecture, computes intermediate triplet messages $\mathbf{m}_{ijk}^{(\tau)}$ that naturally capture the ternary $(i, j) \to k$ structure of edge pointers. However, it immediately pools them via $\max_k$ to produce an edge latent $\mathbf{h}_{ij}^{(\tau)}$, after which the triplet messages are discarded. This lossy triplet-to-edge aggregation at the boundary between processor and decoder mirrors the encoder collapse, and likewise degrades the network's ability to reason about edge-pointer probes (Section 3.2).

We fix both bottlenecks with targeted, lightweight additions to the encoder, processor, and decoder. Our encoder retains the native ternary structure of the pointer tensor by projecting it directly into a low-width triplet feature stream rather than marginalising over axes. We extend the processor to maintain recurrent triplet states that persist across steps, and augment the decoder to score pointer candidates from these states alongside the standard node and edge latents. All additions operate at the narrow triplet dimension ($d_t = 8$), leaving the processor's core reasoning mechanism unchanged and keeping our model lightweight – in the spirit of the encode-process-decode design principle that encoders and decoders should remain simple and entrust the bulk of the reasoning to the processor.

We evaluate our approach on all three CLRS-30 algorithms whose hints and outputs include edge-pointer probes: Floyd–Warshall, Matrix Chain Order, and Optimal BST. Our best configuration achieves a mean OOD accuracy of $\mathbf{91.14}\%$, compared to 69.34% for the Triplet-GMPNN baseline — a gain of over 20 percentage points with no change to the processor architecture. On Floyd–Warshall alone, the score rises from 46% to 95%. Our model achieves the highest overall average accuracy among all compared methods, including much larger models such as FloydNet (Yu et al., 2026), which uses hundreds of millions of parameters.

2. Background

2.1. CLRS-30

CLRS-30 (Veličković et al., 2022) is the standard neural algorithmic reasoning benchmark: thirty classical algorithms, each exposed together with reference execution trajectories that a learned model is asked to reproduce. Every piece of information in a trajectory is represented as a *probe*, described by three tags $(\sigma, \lambda, \varphi)$: its *stage* $\sigma \in \{\text{input}, \text{output}, \text{hint}\}$, its *location* $\lambda \in \{\text{node}, \text{edge}, \text{graph}\}$, and its *datatype* φ (scalar, categorical, mask, pointer, ...). Inputs and outputs are single snapshots of the algorithm's state, while hints form a time series of intermediate states, which exposes *how* the target algorithm progresses and serves as a supervision signal.

Encode-process-decode. Neural algorithmic reasoning models follow the encode-process-decode paradigm (Veličković et al., 2022; Ibarz et al., 2022). Fix a target algorithm $\mathcal{A}$ and a latent width d_h (by default $d_h = 128$). The pipeline is a composition of three stages. *Encoders* are a family $\{\text{enc}_{\sigma,\lambda,\varphi}^{\mathcal{A}}\}$ of learned maps, one per probe signature: each takes a probe value and returns a vector in $\mathbb{R}^{d_h}$. Encoded probes are summed per location to produce the current encoded inputs $\mathbf{x}_i^{(\tau)} \in \mathbb{R}^{d_h}$, $\mathbf{e}_{ij}^{(\tau)} \in \mathbb{R}^{d_h}$, and $\mathbf{g}^{(\tau)} \in \mathbb{R}^{d_h}$, where $i, j \in \{1, \ldots, n\}$ index vertices and τ is the processor timestep. The node latents $\mathbf{h}_i^{(\tau)} \in \mathbb{R}^{d_h}$ of the *processor* are updated as:

$$P \;:\; \big(\mathbf{x}_i^{(\tau)}, \mathbf{h}_i^{(\tau-1)}, \mathbf{e}_{ij}^{(\tau)}, \mathbf{g}^{(\tau)}\big) \longmapsto \mathbf{h}_i^{(\tau)} \tag{1}$$

with $\mathbf{h}_i^{(0)} = \mathbf{0}$. Finally, *decoders* $\{\text{dec}_{\sigma,\lambda,\varphi}^{\mathcal{A}}\}$ emit a prediction for supervised probes from step-τ quantities (output probes use last step embeddings), using linear heads for scalar/mask/categorical probes and pairwise scores over candidate nodes for pointer probes.

Triplet-GMPNN. The base architecture we build on is the *Triplet-GMPNN* processor of (Ibarz et al., 2022), which instantiates P in (1) as a message-passing neural network augmented with a triplet-reasoning branch. Letting $\mathbf{z}_i^{(\tau)} = \mathbf{x}_i^{(\tau)} \,\|\, \mathbf{h}_i^{(\tau-1)}$, the MPNN update is

$$\mathbf{m}_i^{(\tau)} \;=\; \max_{1 \le j \le n} f_m\big(\mathbf{z}_i^{(\tau)}, \mathbf{z}_j^{(\tau)}, \mathbf{e}_{ij}^{(\tau)}, \mathbf{g}^{(\tau)}\big) \tag{2}$$

$$\mathbf{h}_i^{(\tau)} \;=\; f_r\big(\mathbf{z}_i^{(\tau)}, \mathbf{m}_i^{(\tau)}\big) \tag{3}$$

over a fully-connected graph, with f_m, f_r learned MLPs. A per-node gating mechanism is additionally applied on top of $\mathbf{h}_i^{(\tau)}$ to selectively update only those states that need to change.

The defining ingredient of Triplet-GMPNN is a lightweight *triplet-reasoning* branch: dynamic-programming recurrences such as $D_{ij} \leftarrow \bigoplus_k (D_{ik} \otimes D_{kj})$ aggregate over an intermediate node k, which is naturally expressed by first computing a representation per triplet of nodes and then reducing over k to obtain an edge latent,

$$\mathbf{m}_{ijk}^{(\tau)} \;=\; \psi_t\big(\mathbf{z}_i^{(\tau)}, \mathbf{z}_j^{(\tau)}, \mathbf{z}_k^{(\tau)}, \mathbf{e}_{ij}^{(\tau)}, \mathbf{e}_{ik}^{(\tau)}, \mathbf{e}_{kj}^{(\tau)}, \mathbf{g}^{(\tau)}\big) \tag{4}$$

$$\mathbf{h}_{ij}^{(\tau)} \;=\; \phi_t\Big(\max_{1 \le k \le n} \mathbf{m}_{ijk}^{(\tau)}\Big) \tag{5}$$

where ψ_t produces a low-width triplet vector of dimension d_t (by default $d_t = 8$) and ϕ_t lifts the aggregated edge latent

back to the hidden width d_h. $\mathbf{m}_{ijk}^{(\tau)}$ is an internal quantity of step τ. The triplet-GMPNN processor is therefore a function

$$P \;:\; \left(\mathbf{x}_i^{(\tau)},\, \mathbf{h}_i^{(\tau-1)},\, \mathbf{e}_{ij}^{(\tau)},\, \mathbf{g}^{(\tau)}\right) \longmapsto \left(\mathbf{h}_i^{(\tau)}, \mathbf{h}_{ij}^{(\tau)}\right) \qquad (6)$$

where $\mathbf{h}_{ij}^{(\tau)}$ is consumed by the step-τ decoder and recomputed from scratch at the next step. These design choices will be important in Section 3.2.

2.2. Edge pointers in CLRS

In CLRS probe typing, pointer-valued targets are categorical indices over graph nodes. For an edge-local signal, an edge pointer is formally a mapping $(i, j) \to k$, where $(i, j) \in \mathcal{E}$ is a directed edge and $k \in \mathcal{V}$ is the selected node index. Equivalently, one may represent the same state by a binary tensor:

$$X \in \{0,1\}^{n\times n\times n}, \quad X_{ijk} = 1 \iff (i,j) \to k$$

Formally, pointer validity is one-hot in the third axis: for every ordered pair of nodes (i, j), writing $X_{ij:} \in \{0,1\}^n$ for the slice along destinations (all k),

$$\mathbf{1}^\top X_{ij:} = 1$$

so each edge points to exactly one node.

This representation choice aligns with CLRS design principles: benchmark supervision emphasizes discrete structural targets (such as pointers), which are more stable to evaluate than raw numerical-value outputs across model implementations (Veličković et al., 2022).

2.3. Edge pointers in Floyd–Warshall

Floyd–Warshall computes all-pairs shortest paths through the standard dynamic-programming recurrence, with outer relaxation index $\kappa \in \{1, \ldots, n\}$ (the vertex activated at stage κ in the CLRS hint trajectory). Writing $D_{ij}^{(\kappa)}$ for the shortest-path weight from i to j when paths may use only intermediate vertices in $\{1, \ldots, \kappa\}$,

$$D_{ij}^{(\kappa)} = \min\Big(D_{ij}^{(\kappa-1)},\; D_{i\kappa}^{(\kappa-1)} + D_{\kappa j}^{(\kappa-1)}\Big)$$

For shortest-path style dynamic programming in CLRS, supervision is intentionally structural: besides distances, the task predicts pointers that permit explicit path reconstruction. For Floyd–Warshall, this means predicting a pointer matrix Π that supports path reconstruction, not just distance scoring. Concretely, each ordered pair (i, j) carries a pointer $\Pi_{ij} \in \{1, \ldots, n\}$. Initialization sets $\Pi_{ij} = i$, and whenever a relaxation through κ strictly improves the distance,

$$\Pi_{ij} \leftarrow \Pi_{\kappa j} \qquad \text{if} \qquad D_{i\kappa}^{(\kappa-1)} + D_{\kappa j}^{(\kappa-1)} < D_{ij}^{(\kappa-1)}$$

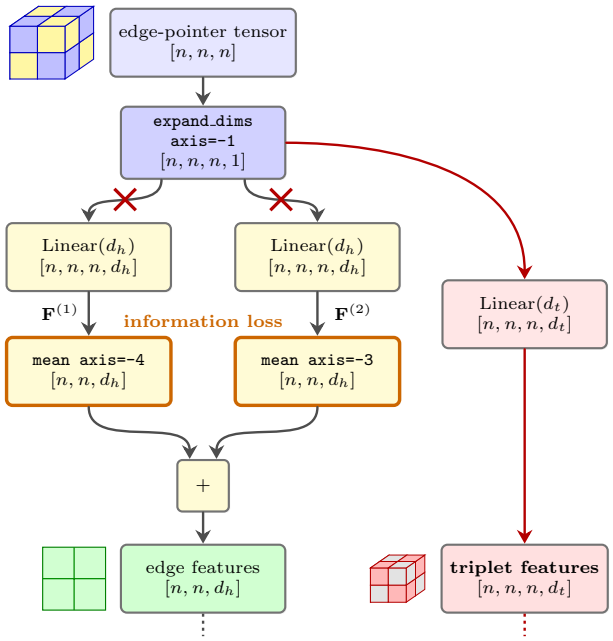

Figure 2. Edge-pointer encoder adaptation. The CLRS-30 encoder applies two shared affine maps to the pointer tensor X, producing $\mathbf{F}^{(1)}$ and $\mathbf{F}^{(2)}$, and then averages them over different axes, collapsing the pointer tensor into a lossy edge feature and thereby causing information loss. Our adaptation instead projects the pointer tensor with a single linear map into a dedicated **triplet features** tensor of width d_t, preserving the native $(i, j) \to k$ structure. Red crosses mark the default connections that are disabled in our setup.

So the model must learn two coupled objects: scalar distance dynamics in D and pointer dynamics in Π. This is exactly where edge pointers matter: for each directed pair (i, j), the network must predict which node index the pair points to in the current algorithmic state.

2.4. Motivation: non-injective edge-pointer encoding

CLRS-30 processors are GNN-based and operate on node, edge, and graph features. To fit this interface, the pointer tensor $X \in \{0,1\}^{n\times n\times n}$, with $X_{ijk} = 1$ iff edge (i, j) points to k, is not kept as a separate higher-order feature but merged into the shared edge slice $\mathbb{R}^{n\times n\times d_h}$. Write $\mathbf{u}_X \in \mathbb{R}^{n\times n\times d_h}$ for its *pointer-only* contribution to those edge features.

The default CLRS edge-pointer encoder (illustrated in Figure 2) applies two learned affine maps entrywise to X, producing $\mathbf{F}_{ijk}^{(s)} \in \mathbb{R}^{d_h}$ for $s \in \{1, 2\}$, and then aggregates them by averaging over different axes of X: the first over the sender axis, the second over the receiver axis. For each directed edge (i, j), we thus have

$$\mathbf{A}_{ij} = \frac{1}{n}\,\mathbf{1}^\top \mathbf{F}_{:,i,j}^{(1)}, \qquad \mathbf{B}_{ij} = \frac{1}{n}\,\mathbf{1}^\top \mathbf{F}_{i,:,j}^{(2)} \qquad (7)$$

and set $(\mathbf{u}_X)_{ij} := \mathbf{A}_{ij} + \mathbf{B}_{ij}$. Since the two pooled maps see only axis-wise averages of X, many distinct tensors X can collapse to the same $\mathbf{u}_X$.

For $n = 2$, fix two tensors $X^{(a)}, X^{(b)} \in \{0,1\}^{2\times2\times2}$, specified via their destination slices $M_\ell^{(r)} := X^{(r)}_{:,:,\ell} \in \{0,1\}^{2\times2}$:

$$M_1^{(a)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad M_2^{(a)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$M_1^{(b)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad M_2^{(b)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{8}$$

Although $X^{(a)} \neq X^{(b)}$, one has $\mathbf{u}_{X^{(a)}} = \mathbf{u}_{X^{(b)}}$ for *every* shared entrywise affine encoding

$$\mathbf{F}^{(s)}_{ijk} = X_{ijk}\, \mathbf{w}^{(s)} + \mathbf{b}^{(s)} \quad s \in \{1,2\}$$

with $\mathbf{w}^{(s)}, \mathbf{b}^{(s)} \in \mathbb{R}^{d_h}$ independent of coordinates — exactly as in the reference CLRS edge-pointer encoder. Figure 1 illustrates the collision in (8), while Appendix C provides the proof details. This is precisely why we move to triplet-level encodings in Section 3.1.

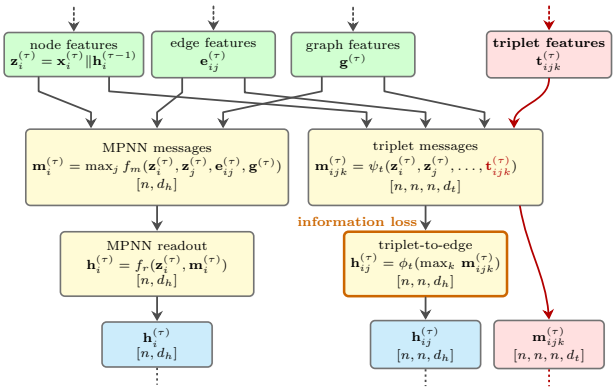

Figure 3. Triplet-GMPNN processor with our triplet-feature extension. The triplet-reasoning branch is already part of the standard Triplet-GMPNN processor. We feed the current triplet features $\mathbf{t}_{ijk}^{(\tau)}$ into this branch, compute internal messages $\mathbf{m}_{ijk}^{(\tau)}$, and expose $\mathbf{m}_{ijk}^{(\tau)}$ as the triplet output state. This avoids the lossy triplet-to-edge aggregation, preserving the ternary pointer signal for downstream components.

3. Fixing Edge-Pointer Compositionality with Triplet Features

3.1. Injective edge-pointer encoding through triplet features

To avoid the marginal collapse of Section 2.4, we explicitly encode edge-pointer information into a recurrent triplet stream. The main idea is simple: keep the native ternary structure $(i,j) \mapsto k$ at triplet level, i.e., maintain triplet states

$$\mathbf{t}_{ijk}^{(\tau)} \in \mathbb{R}^{d_t} \qquad (i,j,k \in \{1,\ldots,n\})$$

instead of projecting to lower-order marginals as the default CLRS-30 edge-pointer encoder does through (7). Figure 2 summarizes the encoder interface. Concretely, our edge-pointer encoder replaces the two pooled maps of (7) by a single entrywise affine map that preserves the triplet axis,

$$\mathbf{t}_{ijk}^{(\tau)} = X_{ijk}^{(\tau)}\, \mathbf{w} + \mathbf{b} \qquad \mathbf{w}, \mathbf{b} \in \mathbb{R}^{d_t} \tag{9}$$

with no aggregation over any axis. This fits naturally with how Triplet-GMPNN already works. The internal triplet messages $\mathbf{m}_{ijk}^{(\tau)}$ of (4) are lightweight (width d_t, with default $d_t = 8$) and are only later merged into hidden-width edge representations through $\phi_t(\max_k \cdot)$. We match the initialization in (9) to this native triplet width, so no extra up- or down-projection is needed.

3.2. Triplet-GMPNN processor adaptation

We keep the processor structure lightweight and only extend the triplet pathway needed for injective edge-pointer reasoning: we feed current-step triplet states $\mathbf{t}_{ijk}^{(\tau)}$ into the triplet-reasoning branch, and keep a dedicated triplet output state $\mathbf{m}_{ijk}^{(\tau)}$ instead of discarding the computed triplet messages after pooling.

Triplet messages with recurrent triplet input. We replace the baseline triplet-message function ψ_t from (4) with an extended variant $\widetilde{\psi}_t$ that takes previous-step triplet states as an additional argument while keeping the existing signature for the node, edge, and graph latents:

$$\begin{aligned} \mathbf{m}_{ijk}^{(\tau)} = \widetilde{\psi}_t\big(&\mathbf{z}_i^{(\tau)}, \mathbf{z}_j^{(\tau)}, \mathbf{z}_k^{(\tau)}, \\ &\mathbf{e}_{ij}^{(\tau)}, \mathbf{e}_{ik}^{(\tau)}, \mathbf{e}_{kj}^{(\tau)}, \\ &\mathbf{g}^{(\tau)}, \mathbf{t}_{ijk}^{(\tau)}\big) \end{aligned} \tag{10}$$

Because $\mathbf{t}_{ijk}^{(\tau)}$ is already in the native width d_t of the triplet branch, injecting it requires no up- or down-projection: concretely, we implement $\widetilde{\psi}_t$ by concatenating $\mathbf{t}_{ijk}^{(\tau)}$ with the remaining inputs before the triplet MLP.

Preserving triplet messages in the processor output. In the baseline, the triplet messages $\mathbf{m}_{ijk}^{(\tau)}$ exist only as a transient intermediate: they are immediately consumed by $\phi_t(\max_k \cdot)$ in (5) to produce the edge latent $\mathbf{h}_{ij}$, and then discarded. This is a natural triplet-to-edge bottleneck, analogous at the processor level to the marginal collapse of Section 2.4 at the encoder level. We keep the edge latent $\mathbf{h}_{ij}^{(\tau)} = \phi_t(\max_k \mathbf{m}_{ijk}^{(\tau)})$ exactly as in (5), so that the rest of the Triplet-GMPNN pipeline is unchanged, but we additionally expose the triplet messages to the decoder (Section 3.3).

The adapted processor is now the following function:

$$\widetilde{P} : \left(\mathbf{x}_i^{(\tau)}, \mathbf{h}_i^{(\tau-1)}, \mathbf{e}_{ij}^{(\tau)}, \mathbf{g}^{(\tau)}, \mathbf{t}_{ijk}^{(\tau)}\right) \longmapsto \left(\mathbf{h}_i^{(\tau)}, \mathbf{h}_{ij}^{(\tau)}, \mathbf{m}_{ijk}^{(\tau)}\right) \tag{11}$$

so that the ternary pointer signal is preserved across processor steps and remains available to the decoder. Figure 3 illustrates the two pathways.

3.3. Decoder adaptation for edge pointers

Injective encoding is paired with a small decoder extension, illustrated in Figure 4. Fix a directed edge (i, j) and a candidate destination $c \in \{1, \ldots, n\}$ for the pointer on that edge. We write *logits* $L_{ijc}^{(\tau)} \in \mathbb{R}$ for the score of choosing c on (i, j) at processor timestep τ. Stacking all edges and candidates yields a tensor $L^{(\tau)} \in \mathbb{R}^{n\times n\times n}$ with entries $L_{ijc}^{(\tau)}$. The third index c plays the same role as the pointer axis k in the one-hot tensor X_{ijk} from Section 2.4, but we reserve c here so that "L_{ijc}" always denotes a logit and "X_{ijk}" always denotes data.

Baseline edge-pointer logits. The baseline branch follows the standard CLRS pointer decoder for an edge-located probe: it forms an edge summary from the incident node and edge latents, then scores each candidate destination against that summary. Fix τ, and let g be the learned edge-mixing map from the CLRS baseline and let f be its learned pointer readout. We form an edge summary $\mathbf{q}_{ij}^{(\tau)} \in \mathbb{R}^{d_h}$ and a baseline logit for each candidate c by

$$\mathbf{q}_{ij}^{(\tau)} = g\left(\mathbf{x}_i^{(\tau)}, \mathbf{x}_j^{(\tau)}, \mathbf{h}_i^{(\tau)}, \mathbf{h}_j^{(\tau)}, \mathbf{e}_{ij}^{(\tau)}, \mathbf{h}_{ij}^{(\tau)}\right) \tag{12}$$

$$L_{ijc}^{\text{base}} = f\left(\mathbf{q}_{ij}^{(\tau)}, \mathbf{x}_c^{(\tau)}, \mathbf{h}_c^{(\tau)}\right) \tag{13}$$

Triplet logit fusion. We keep that baseline map unchanged and add a term L_{ijc}^{trip} derived directly from the processor triplet messages $\mathbf{m}_{ijc}^{(\tau)}$ and encoded triplet features $\mathbf{t}_{ijc}^{(\tau)}$. Concretely, we use a single linear readout $l_{\text{trip}} : \mathbb{R}^{d_t} \times \mathbb{R}^{d_t} \to \mathbb{R}$,

$$L_{ijc}^{\text{trip}} = l_{\text{trip}}\left(\mathbf{m}_{ijc}^{(\tau)}, \mathbf{t}_{ijc}^{(\tau)}\right) \tag{14}$$

and define the combined logits at timestep τ as the sum of the two pathways,

$$L_{ijc}^{(\tau)} = L_{ijc}^{\text{base}} + L_{ijc}^{\text{trip}} \tag{15}$$

At prediction time, CLRS distinguishes *hard* and *soft* post-processing. Hard decoding picks $\hat{c}_{ij}^{(\tau)} = \arg\max L_{ij,:}^{(\tau)}$. Soft decoding applies $\mathrm{softmax}$ to $L_{ij,:}^{(\tau)}$, yielding a fractional edge pointer that is fed back at timestep $\tau + 1$ when training with relaxed trajectories.

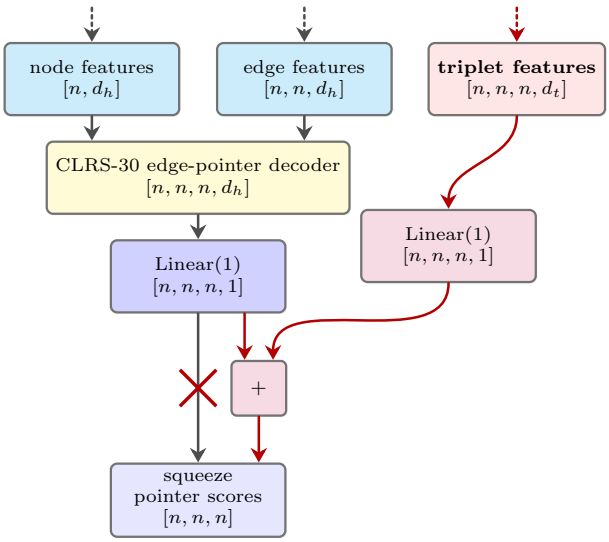

Figure 4. Edge-pointer decoder adaptation. We keep the per-(i, j, k) logits produced by the default CLRS-30 edge-pointer decoder, and add to them a second set of logits computed directly from **triplet features**.

Although the decoder produces logits over all triplets (i, j, c), the hidden representations feeding into it are only node-level ($\mathbb{R}^{n\times d_h}$) and edge-level ($\mathbb{R}^{n\times n\times d_h}$), so no hidden-width triplet tensor needs to be materialised. Naively lifting triplet features to full width d_h instead would introduce an additional dense hidden-width triplet branch, substantially increasing peak activation memory. This is why we keep the added triplet pathway low-width ($d_t = 8$ by default) and fuse it additively at the logit level.

4. Related Work

Processor-focused NAR optimisations. Following the CLRS-30 benchmark (Veličković et al., 2022) and the Triplet-GMPNN baseline (Ibarz et al., 2022), the dominant direction of NAR research has been improving the *processor*. Works have proposed attention-based and transformer-style processors (Diao & Loynd, 2022; Müller et al., 2024; Yu et al., 2026), context-aware message-passing (Shi et al., 2024), Markov-property-preserving architectures (Bohde et al., 2024), richer node and edge representations (Mahdavi et al., 2022), recurrent aggregation (Xu & Veličković, 2024), and neural priority-queue structures (Jain et al., 2023). Some of this work draws on deep mathematical connections: Dudzik & Veličković (2022) ground triplet reasoning in the categorical semantics of dynamic programming, while Hashemi et al. (2025) leverage tropical geometry to design attention mechanisms tailored to combinatorial algorithms. Požgaj et al. (2025) push beyond polynomial-time algorithms, proposing a two-stage pipeline that teaches

NAR models to tackle pseudo-polynomial problems. These processor improvements occasionally come at a cost: the resulting architectures have higher time and memory complexity than their respective baselines.

Several recent models move beyond node-only message passing in ways that are partially related to Triplet-GMPNN: TEA (Jung & Ahn, 2023) introduces triplet-informed edge latents, Edge Transformer (Müller et al., 2024) operates on node-pair tokens with pair-level relational updates, and FloydNet (Yu et al., 2026) iteratively refines a global all-pairs relation tensor. Their higher-order processors are naturally favorable for triplet-based algorithmic reasoning, which makes the edge-pointer encoder-decoder bottleneck we identify in the standard CLRS pipeline even more pronounced.

Training- and data-focused NAR optimisations. A complementary line of work improves NAR by modifying the training procedure or the information available to the model rather than the architecture itself. Open-book NAR (Li et al., 2024) augments the model with access to stored trajectories at inference time. Georgiev et al. (2024) train models to reach a fixed-point equilibrium, replacing the step-by-step hint supervision with an implicit depth objective that improves generalisation.

Encoder-focused NAR optimisations. To the best of our knowledge, the approach of Huang et al. (2026) is the only prior NAR work that explicitly targets the encoder. They use a more expressive encoder together with an auxiliary reconstruction objective that reconstructs the input state from encoded representations. In contrast, we follow the original NAR design principle that encoder/decoder modules remain simple while most algorithmic logic is placed in the processor. This separation supports stronger multi-task sharing, easier transfer to new settings, and lower risk of task-specific overfitting.

5. Experiments

5.1. Evaluation setup

Floyd–Warshall, Matrix Chain Order, and Optimal BST are the CLRS-30 tasks in which some intermediate hints, as well as the algorithm's output, are represented with edge-pointer probes. We therefore limit our experiments to these three tasks. We fully follow the CLRS-30 evaluation protocol and inherit all hyperparameters from CLRS-30 unless stated otherwise. We report means and standard deviations over three random seeds. Models are trained on samples with sizes $n \leq 16$. During training, we periodically evaluate on in-distribution samples of size $n = 16$, and use the checkpoint with the best in-distribution score so far for out-of-distribution evaluation on samples of size $n = 64$. In all our experiments, the default CLRS-30 model and the base architecture for our upgrades is Triplet-GMPNN.

5.2. Experiment groups

Edge hidden states. Triplet-GMPNN maintains recurrent hidden states and uses a gating mechanism, motivated by the fact that many algorithms update only a small number of states at each step. The default Triplet-GMPNN carries only node states $\mathbf{h}_i^{(\tau)}$ as in (1); the per-step edge latent $\mathbf{h}_{ij}^{(\tau)}$ from the triplet branch of (5) is handed to the decoder of step τ and then recomputed at step $\tau + 1$. In our setting, retaining an explicit **edge hidden state** is a natural extension because the key quantities for our three dynamic-programming tasks are carried by edge features. Similarly to how we encode node features (i.e., $\mathbf{z}_i^{(\tau)} = \mathbf{x}_i^{(\tau)} \| \mathbf{h}_i^{(\tau-1)}$), in equations (4) and (10) we replace edge features $\mathbf{e}_{ij}^{(\tau)}$ with $\mathbf{z}_{ij}^{(\tau)} = \mathbf{h}_{ij}^{(\tau-1)} \| \mathbf{e}_{ij}^{(\tau)}$. In other words, the processor's recurrent state becomes $(\mathbf{h}_i^{(\tau)}, \mathbf{h}_{ij}^{(\tau)})$. Consequently, we also add an additional gate mechanism for $\mathbf{h}_{ij}^{(\tau)}$ to match the node hidden-state gating mechanism.

Edge-level relational representations are not a new ingredient in NAR. Prior works have used edge latents or pair-level states in different forms (Jung & Ahn, 2023; Shi et al., 2024; Müller et al., 2024; Yu et al., 2026). We use this ablation to decide whether to keep edge hidden states enabled, and we keep the chosen setting fixed for all subsequent experiments unless otherwise stated.

Edge-pointer scheme comparison. We compare the standard CLRS-30 edge-pointer encoding-decoding scheme against our triplet-feature scheme while keeping the rest of the Triplet-GMPNN pipeline aligned. The goal is to isolate how much triplet features help the baseline Triplet-GMPNN model on the three edge-pointer tasks. To make this effect explicit, we run the comparison in two variants: without edge hidden states and with edge hidden states enabled. This lets us assess whether gains from triplet features are consistent across both hidden-state settings, rather than being tied to only one particular configuration.

Table 1. Edge-pointer scheme and edge-hidden-state ablation. OOD average accuracy at $n = 64$. The first row is the default baseline: Triplet-GMPNN with the standard CLRS-30 edge-pointer encoder-decoder, without triplet features or edge hiddens.

TripletFts	Edge hid.	Floyd–Warshall	Matrix Chain Order	Optimal BST	Avg.
×	×	46.00% ± 0.69	90.71% ± 1.08	71.30% ± 9.54	69.34%
×	✓	59.90% ± 9.74	93.55% ± 1.49	**80.74**% ± 0.43	78.07%
✓	×	85.24% ± 5.14	98.04% ± 0.41	78.74% ± 3.72	87.34%
✓	✓	**95.00**% ± 0.40	**98.08**% ± 1.12	80.36% ± 1.89	**91.14**%

Table 2. OOD average accuracy scores on the three edge-pointer CLRS-30 tasks. Methods marked with * use substantially larger models (parameter count) than ours. In each task column, bold denotes methods whose reported uncertainty interval overlaps that of the top entry.

Method	Floyd–Warshall	Matrix Chain Order	Optimal BST	Avg.
TripletFts-GMPNN (ours)	**95.00**% ± 0.40	**98.08**% ± 1.12	80.36% ± 1.89	**91.14**%
FloydNet (no-hint)* (Yu et al., 2026)	**96.20**% ± 1.93	85.15% ± 0.00	74.50% ± 0.00	85.28%
ReNAR (Huang et al., 2026)	71.53% ± 4.64	94.13% ± 2.21	**83.23**% ± 2.43	82.96%
ET (Müller et al., 2024)	83.57% ± 1.79	90.11% ± 3.28	71.70% ± 5.46	81.79%
FloydNet (hint)* (Yu et al., 2026)	64.20% ± 0.00	91.50% ± 1.41	**82.87**% ± 0.96	79.52%
G-ForgetNet (Bohde et al., 2024)	56.32% ± 0.86	91.08% ± 0.51	**83.58**% ± 0.49	76.99%
CEF-GMPNN (Shi et al., 2024)	55.42% ± 2.58	92.96% ± 0.98	79.43% ± 2.99	75.94%
Open-Book NAR (Li et al., 2024)	51.52% ± 1.74	90.85% ± 1.11	70.04% ± 2.38	70.80%
Triplet-GMPNN (Ibarz et al., 2022)	46.00% ± 0.69	90.71% ± 1.08	71.30% ± 9.54	69.34%
TEAM (Jung & Ahn, 2023)	35.33% ± 2.82	94.46% ± 0.88	70.70% ± 5.45	66.83%
RT (Diao & Loynd, 2022)	31.59% ± 7.60	91.89% ± 1.20	74.40% ± 2.60	65.96%
PGN (Veličković et al., 2020)	28.76% ± 0.51	83.91% ± 0.49	71.01% ± 1.82	61.23%
MP-2WL (Mahdavi et al., 2022)	33.59% ± 1.36	68.70% ± 3.11	76.12% ± 2.06	59.47%
MPNN (Gilmer et al., 2017)	26.74% ± 1.77	79.84% ± 1.40	62.23% ± 0.44	56.27%
Memnet (Sukhbaatar et al., 2015)	14.17% ± 0.13	81.96% ± 1.03	72.03% ± 1.21	56.05%
GAT (Veličković et al., 2017)	16.66% ± 3.14	78.19% ± 3.31	65.49% ± 1.75	53.45%
Deep Sets (Zaheer et al., 2017)	7.22% ± 0.90	81.96% ± 1.03	69.71% ± 1.36	52.96%
NPQ (Jain et al., 2023)	17.84% ± 1.09	78.74% ± 1.01	59.89% ± 1.84	52.16%
RNAR (Xu & Veličković, 2024)	27.49% ± 6.95	25.12% ± 1.86	36.04% ± 12.55	29.55%

Comparison with prior work. We compare our best configuration with prior methods that report results on these three edge-pointer algorithms. Training setups are not identical across methods, but all reported results follow the CLRS-30 evaluation paradigm for these tasks. Our Triplet-GMPNN results are obtained from our own runs, while results for other methods are taken from the literature. For FloydNet (Yu et al., 2026), standard deviations are not available for some reported entries. To keep a uniform table format across methods, those entries are shown with ± 0.00.

Factorial ablation of triplet features. Introducing our triplet features requires corresponding adaptations in the encoder, processor, and decoder. This allows us to run a full factorial ablation over all 2^3 combinations and examine in more detail which components contribute most to the final gains, and whether improvements depend on all three components or only a subset.

Control setting with ground-truth outputs as an additional input. We run an additional stress test to empirically evaluate whether the default CLRS-30 edge-pointer encoding-decoding scheme loses information. In addition to the usual inputs for each sample, we provide the target algorithm output used for supervision (the exact solution object for that instance). This oracle signal is present in both training and evaluation. If the pipeline preserves information well, this setting should be easy for the model, since it can in principle learn an identity map from the provided target and reach near-perfect scores. Conversely, if the pipeline discards or distorts information, adding the ground-truth output in this form should provide little benefit.

6. Results

Enabling edge hidden states is beneficial and selected for later experiments. In Table 1, enabling edge hidden states helps on every algorithm, both with the standard CLRS-30 edge-pointer scheme and with triplet features. The associated increase in parameter count is modest relative to these gains (see Appendix B).

Triplet-feature edge-pointer scheme yields large OOD gains. Table 1 shows that replacing the standard CLRS-30 edge-pointer scheme with our triplet-feature design sharply lifts the average accuracy scores, independently of whether edge hidden states are used. For Floyd–Warshall, OOD average accuracy achieves 95.00%, a large margin over the default CLRS-30 edge-pointer scheme on this task – the case that motivated this work and framed its central objective. For Matrix Chain Order, OOD average accuracy reaches 98.08% at $n = 64$, largely closing the remaining gap to near-saturated performance at that evaluation scale. For Optimal BST, even the strongest configuration sits near 80% accuracy, well below the other two edge-pointer tasks. This suggests that algorithm-specific structure still limits performance and raises questions for future work.

Our model ranks first in the direct external comparison. Table 2 shows that our best model has the highest overall average (91.14%), and achieves the best performance on Matrix Chain Order. On Floyd–Warshall, our model substantially outperforms all methods except FloydNet (no-hint). This comparison should be interpreted with care, as FloydNet uses substantially larger models than ours (see Appendix B). Importantly, the bottleneck we address is

Table 3. Full-factorial ablation of triplet features over the encoder, processor, and decoder ($E/P/D$). ✓/× indicate whether the triplet pathway is enabled in each stage.

TripletFts			Floyd–Warshall	Matrix Chain Order	Optimal BST	Avg.
E	P	D				
×	×	×	59.90% ± 9.74	93.55% ± 1.49	80.74% ± 0.43	78.07%
×	×	✓	61.16% ± 9.68	88.29% ± 7.38	79.18% ± 1.59	76.21%
×	✓	×	64.13% ± 9.39	84.73% ± 6.92	78.12% ± 2.88	75.66%
×	✓	✓	**95.36**% ± 0.12	**98.26**% ± 0.67	81.16% ± 1.70	**91.59**%
✓	×	×	59.57% ± 8.36	90.66% ± 2.94	78.72% ± 0.89	76.32%
✓	×	✓	59.09% ± 10.64	92.17% ± 1.11	79.19% ± 1.16	76.81%
✓	✓	×	66.64% ± 6.60	94.13% ± 2.44	80.83% ± 1.30	80.53%
✓	✓	✓	**95.00**% ± 0.40	**98.08**% ± 1.12	80.36% ± 1.89	**91.14**%

Table 4. Diagnostic stress test. *Oracle*: the ground-truth output is (✓) or is not (×) provided as an additional input in training and evaluation.

Oracle	TripletFts	Floyd–Warshall	Matrix Chain Order	Optimal BST	Avg.
×	×	59.90% ± 9.74	93.55% ± 1.49	80.74% ± 0.43	78.07%
×	✓	95.00% ± 0.40	98.08% ± 1.12	80.36% ± 1.89	91.14%
✓	×	62.67% ± 5.12	92.43% ± 0.87	78.04% ± 5.71	77.71%
✓	✓	**99.99**% ± 0.01	**99.68**% ± 0.43	**99.54**% ± 0.48	**99.74**%

largely orthogonal to the manner in which prior work improves NAR models. Other architectures could benefit from our triplet features, and we expect that such a combination would push the scores even higher.

Processor and decoder triplet pathways are sufficient for top performance. Table 3 shows that $(\times, \checkmark, \checkmark)$ and $(\checkmark, \checkmark, \checkmark)$ in the $E/P/D$ columns are the two strongest configurations in this ablation. This indicates that preserving triplet information through processing and decoding is more important than injecting it at the encoder. This is because edge-pointer probes in these dynamic-programming algorithms reflect the *output* – backtracking pointers used to reconstruct the solution – whereas the core recurrence is driven by the dynamic programming table. For example, in Floyd–Warshall, updates to the distance matrix D carry the main transition, while Π-style pointers are used primarily for reconstruction.

The CLRS-30 edge-pointer encoder-decoder scheme causes severe information loss. Table 4 reports model performance when the ground-truth output is provided as an additional input during training and evaluation. Under the standard CLRS-30 edge-pointer encoder-decoder, the oracle signal does not improve performance at all. That suggests the default scheme is not able to work with this type of data, supporting our initial hypothesis of an information bottleneck in the CLRS-30 edge-pointer encoder-decoder. By contrast, with triplet-based encoding and decoding, the model can exploit the oracle and reaches near-perfect scores on all three tasks.

7. Conclusion

We identified two coupled information bottlenecks in the standard CLRS-30 encode-process-decode pipeline for edge-pointer probes. The first is a provably non-injective edge-pointer encoder that collapses distinct pointer states to identical edge embeddings. The second is a lossy triplet-to-edge aggregation inside the Triplet-GMPNN processor that discards ternary pointer structure before it reaches the decoder. We addressed both with lightweight, targeted modifications – an injective triplet-level encoder, a recurrent triplet state in the processor, and a triplet-augmented decoder – all operating at the narrow width $d_t = 8$ and leaving the processor's core reasoning mechanism unchanged. On the three CLRS-30 algorithms that include edge-pointer probes, our model achieves a mean OOD accuracy of 91.14% and ranks first among all compared methods including larger, more complex models. Our results underscore that meaningful gains can also come from scrutinising the encoding-decoding pipeline rather than scaling up the processor – a complementary and largely overlooked direction in the NAR literature.

Limitations and Future Work. Our modifications are specifically designed for probes of the edge-pointer type, particularly the three CLRS-30 algorithms that use such probes (Floyd–Warshall, Matrix Chain Order, and Optimal BST). Extending the approach to the remaining 27 algorithms in CLRS-30, which do not use edge-pointer probes, is ongoing work. More broadly, it is an open question whether recurrent triplet features – which here serve as the natural carrier for the ternary edge-pointer structure – can be repurposed to benefit algorithms that use only node pointers or other probe types, where the encoding-decoding scheme does not exhibit the same injectivity failure. Furthermore, since our encoder-decoder fix is orthogonal to processor-level improvements, combining it with other processors (Müller et al., 2024; Bohde et al., 2024; Yu et al., 2026) is a natural next step, and we expect those models to similarly benefit from resolving the edge-pointer bottleneck. Finally, Optimal BST remains noticeably harder than the other two tasks even after our improvements, suggesting that algorithm-specific structure beyond the encoding-decoding pipeline plays a role, and warrants further investigation.

Acknowledgements

The authors acknowledge the support of the Croatian Science Foundation through research project *Neural Combinatorial Optimization for Software Defect Prediction* (IP-2025-02-8486).

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Bohde, M., Liu, M., Saxton, A., and Ji, S. On the markov property of neural algorithmic reasoning: Analyses and methods. *arXiv preprint arXiv:2403.04929*, 2024.

Diao, C. and Loynd, R. Relational attention: Generalizing transformers for graph-structured tasks. *arXiv preprint arXiv:2210.05062*, 2022.

Dudzik, A. J. and Veličković, P. Graph neural networks are dynamic programmers. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=wu1Za9dY1GY.

Georgiev, D. G., Wilson, J., Buffelli, D., and Lio, P. Deep equilibrium algorithmic reasoning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=SuLxkxCENa.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. Pmlr, 2017.

Hashemi, B., Pasque, K., Teska, C., and Yoshida, R. Tropical attention: Neural algorithmic reasoning for combinatorial algorithms. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

He, Y., Nerem, R. R., Stoll, T., Cantürk, S., Georgiev, D., Qian, C., Wittig, S., Geerts, F., Jegelka, S., Vitercik, E., Wang, Y., Karalias, N., and Morris, C. Position: Neural algorithmic reasoning requires a clear scope, theoretical foundations, and empirical rigor. *arXiv preprint*, 2026.

Huang, J., Peng, C., Xu, C., Yang, Z., Cai, K., Zhang, C., Wang, Y., Gong, Y., Zhou, W., and Zheng, I. Richer representations for neural algorithmic reasoning via auxiliary reconstruction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 21966–21974, 2026.

Ibarz, B., Kurin, V., Papamakarios, G., Nikiforou, K., Bennani, M., Csordás, R., Dudzik, A. J., Bošnjak, M., Vitvitskyi, A., Rubanova, Y., et al. A generalist neural algorithmic learner. In *Learning on graphs conference*, pp. 2–1. PMLR, 2022.

Jain, R., Veličković, P., and Liò, P. Neural priority queues for graph neural networks. *arXiv preprint arXiv:2307.09660*, 2023.

Jung, Y. and Ahn, S. Triplet edge attention for algorithmic reasoning. In *The Second Learning on Graphs Conference*, 2023.

Li, H., Peng, C., Xu, C., and Yang, Z. Open-book neural algorithmic reasoning. *Advances in Neural Information Processing Systems*, 37:9819–9836, 2024.

Mahdavi, S., Swersky, K., Kipf, T., Hashemi, M., Thrampoulidis, C., and Liao, R. Towards better out-of-distribution generalization of neural algorithmic reasoning tasks. *arXiv preprint arXiv:2211.00692*, 2022.

Müller, L., Kusuma, D., Bonet, B., and Morris, C. Towards principled graph transformers. *Advances in Neural Information Processing Systems*, 37:126767–126801, 2024.

Požgaj, S., Georgiev, D. G., Šilić, M., and Veličković, P. KNARsack: Teaching neural algorithmic reasoners to solve pseudo-polynomial problems. In *The Fourth Learning on Graphs Conference*, 2025. URL https://openreview.net/forum?id=cwj418JPv7.

Shi, S., Peng, C., Xu, C., and Yang, Z. A context-enhanced framework for sequential graph reasoning. *arXiv preprint arXiv:2412.09056*, 2024.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Veličković, P., Buesing, L., Overlan, M., Pascanu, R., Vinyals, O., and Blundell, C. Pointer graph networks. *Advances in neural information processing systems*, 33: 2232–2244, 2020.

Veličković, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pp. 22084–22102. PMLR, 2022.

Xu, K. and Veličković, P. Recurrent aggregators in neural algorithmic reasoning. *arXiv preprint arXiv:2409.07154*, 2024.

Xu, K., Zhang, M., Li, J., Du, S. S., Kawarabayashi, K., and Jegelka, S. How neural networks extrapolate: From feedforward to graph neural networks. In *9th International Conference on Learning Representations, ICLR*

2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021. URL https://openreview.net/forum?id=UH-cmocLJC.

Yu, J., Zeng, M., and Ye, Q. Floydnet: A learning paradigm for global relational reasoning. *arXiv preprint arXiv:2601.19094*, 2026.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. Deep sets. corr abs/1703.06114 (2017). *arXiv preprint arXiv:1703.06114*, 2017.

A. Validation Results

Table 5 reports in-distribution validation scores ($n = 16$) for the same two-factor layout as Table 1. We observe that, in this case as well, both edge hidden states and triplet features improve the model's performance.

Table 5. In-distribution validation.

TripletFts	Edge hid.	Floyd–Warshall	Matrix Chain Order	Optimal BST	Avg.
×	×	93.78% ± 1.07	99.62% ± 0.10	97.65% ± 0.12	97.02%
×	✓	96.53% ± 1.40	99.72% ± 0.08	97.94% ± 0.16	98.06%
✓	×	96.12% ± 1.30	99.58% ± 0.05	97.65% ± 0.12	97.78%
✓	✓	99.24% ± 0.06	99.69% ± 0.08	98.64% ± 0.06	99.19%

B. Additional Experimental Details

All experiments were performed on a single NVIDIA A100 GPU with 40 GB of memory. To estimate memory usage, we disabled JAX GPU preallocation and tracked peak GPU memory allocation, observing ~ 13.7 GB across all model/task combinations in Table 1. Table 6 lists total model parameters. Hidden edge states increase the total parameter count by about 22%, a modest overhead relative to the generalization gains in Table 1. Triplet features yield a slightly *lower* total parameter count. The CLRS-30 edge-pointer encoder represents pointer probes with 128-dimensional embeddings, whereas the injective triplet encoding uses a width of $d_t = 8$ (Section 3.1). The net effect is a small reduction in total parameters rather than an increase, despite modest additional parameters in the processor and decoder. Table 7 reports wall-clock training time in minutes. Runtime is largely determined by the task algorithm, while edge hidden states and triplet features change it only modestly.

For context, FloydNet (Yu et al., 2026) is not directly comparable to our CLRS setting because its two reported variants differ from ours along several axes at once. Their model depth L is chosen separately for each variant: the hint-based variant is limited to $L = 8$ (reportedly due to memory), while the no-hint variant scales up to $L = 80$. This translates into very different parameter counts: they report roughly 20.2M parameters for the hint variant and 215.0M for the no-hint variant, which are about $27\times$ and $283\times$ larger than our best model (~ 761K parameters, Table 6), respectively. They also report substantially longer training, with up to 80K training steps versus the standard 10K CLRS setup. Beyond scale, their no-hint regime is not just "hints turned off" in the standard recurrent CLRS rollout: when training without hints they compress trajectories into a single step, whereas the classic CLRS setup (and ours) remains step-wise and recurrent over the algorithm trajectory. This single-step training setup limits test-time scaling over longer trajectories, so it is not an equivalent variant of the same recurrent protocol.

Table 6. Total number of model parameters.

TripletFts	Edge hid.	Floyd–Warshall	Matrix Chain Order	Optimal BST
×	×	625,089	624,448	625,987
×	✓	761,281	760,640	762,307
✓	×	624,843	624,202	625,741
✓	✓	761,035	760,394	762,061

Table 7. Wall-clock training time in minutes (one random seed).

TripletFts	Edge hid.	Floyd–Warshall	Matrix Chain Order	Optimal BST
×	×	20.7	25.0	18.0
×	✓	21.5	26.3	20.2
✓	×	21.2	25.7	17.8
✓	✓	22.8	26.8	19.6

C. Edge-pointer encoder collisions

C.1. Collision at $n = 2$

Recall from (8) the two tensors $X^{(a)}, X^{(b)} \in \{0,1\}^{2\times 2\times 2}$, specified through their destination slices $M_\ell^{(r)} := X^{(r)}_{:,:,\ell} \in \{0,1\}^{2\times 2}$:

$$M_1^{(a)} = I \quad M_2^{(a)} = J \qquad M_1^{(b)} = J \quad M_2^{(b)} = I$$

with $I = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$ and $J = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. The two tensors thus swap the roles of identity and off-diagonal between the two destinations $\ell \in \{1,2\}$, in particular $X^{(a)} \neq X^{(b)}$.

Permutation-matrix identities. Both I and J are 2×2 permutation matrices, so every row and every column of each $M_\ell^{(r)}$ sums to 1. Translating back through $M_\ell^{(r)} = X^{(r)}_{:,:,\ell}$, for every $(i,j) \in \{1,2\}^2$ and every $r \in \{a,b\}$,

$$\mathbf{1}^\top X^{(r)}_{:,i,j} \;=\; 1 \qquad \mathbf{1}^\top X^{(r)}_{i,:,j} \;=\; 1 \tag{16}$$

Encoder collision. The reference CLRS edge-pointer encoder uses shared entrywise affine maps $\mathbf{F}^{(s)}_{ijk} = X_{ijk}\,\mathbf{w}^{(s)} + \mathbf{b}^{(s)}$ with $\mathbf{w}^{(s)}, \mathbf{b}^{(s)} \in \mathbb{R}^{d_h}$ independent of (i,j,k). Substituting this form into (7) gives

$$\mathbf{A}_{ij} = \frac{1}{n}\,\mathbf{1}^\top\big(X_{:,i,j}\,\mathbf{w}^{(1)} + \mathbf{b}^{(1)}\big) \qquad \mathbf{B}_{ij} = \frac{1}{n}\,\mathbf{1}^\top\big(X_{i,:,j}\,\mathbf{w}^{(2)} + \mathbf{b}^{(2)}\big)$$

Since $\mathbf{w}^{(s)}$ and $\mathbf{b}^{(s)}$ are constant over the averaged axis and $\mathbf{1}^\top\mathbf{1} = n$, the bias terms simplify to $\mathbf{b}^{(s)}$ and we get

$$\mathbf{A}_{ij} = \frac{1}{n}\big(\mathbf{1}^\top X_{:,i,j}\big)\,\mathbf{w}^{(1)} + \mathbf{b}^{(1)} \qquad \mathbf{B}_{ij} = \frac{1}{n}\big(\mathbf{1}^\top X_{i,:,j}\big)\,\mathbf{w}^{(2)} + \mathbf{b}^{(2)}$$

By (16), at $n = 2$ this yields, for both $r \in \{a,b\}$ and every $(i,j) \in \{1,2\}^2$,

$$\mathbf{A}^{(r)}_{ij} = \tfrac{1}{2}\,\mathbf{w}^{(1)} + \mathbf{b}^{(1)} \qquad \mathbf{B}^{(r)}_{ij} = \tfrac{1}{2}\,\mathbf{w}^{(2)} + \mathbf{b}^{(2)}$$

which is independent of both (i,j) and r. Written over the edge grid $\{1,2\}^2$, both $\mathbf{A}^{(a)}$ and $\mathbf{A}^{(b)}$ are the constant 2×2 tile

$$\mathbf{A}^{(r)} = \begin{pmatrix} \frac{1}{2}\mathbf{w}^{(1)} + \mathbf{b}^{(1)} & \frac{1}{2}\mathbf{w}^{(1)} + \mathbf{b}^{(1)} \\ \frac{1}{2}\mathbf{w}^{(1)} + \mathbf{b}^{(1)} & \frac{1}{2}\mathbf{w}^{(1)} + \mathbf{b}^{(1)} \end{pmatrix}$$

and analogously for $\mathbf{B}^{(r)}$. Stacking over edges, the pointer-only edge-feature block $\mathbf{u}_{X^{(r)}} = \mathbf{A}^{(r)} + \mathbf{B}^{(r)}$ has every entry equal to $\frac{1}{2}(\mathbf{w}^{(1)} + \mathbf{w}^{(2)}) + \mathbf{b}^{(1)} + \mathbf{b}^{(2)}$. In particular $\mathbf{u}_{X^{(a)}} = \mathbf{u}_{X^{(b)}}$, while $X^{(a)} \neq X^{(b)}$.

C.2. Collision at $n \geq 3$

We extend the construction of the previous subsection to arbitrary $n \geq 3$ by "padding" the two $n = 2$ tensors with a fixed common pattern that makes every edge outside the top-left 2×2 corner point to node n. To keep the notation unambiguous, denote by $Y^{(a)}, Y^{(b)} \in \{0,1\}^{2\times 2\times 2}$ the two tensors of the previous subsection (denoted $X^{(a)}, X^{(b)}$ there); inside this subsection, the symbols $X^{(a)}, X^{(b)}$ refer to the new $n \times n \times n$ tensors defined below. For $n \geq 3$, put

$$X^{(r)}_{ijk} := \begin{cases} Y^{(r)}_{ijk} & \text{if } i,j,k \leq 2, \\ 1 & \text{if } (i > 2 \text{ or } j > 2) \text{ and } k = n, \\ 0 & \text{otherwise,} \end{cases} \qquad r \in \{a,b\}$$

Equivalently, every directed edge $(i,j) \in [n]^2$ is routed as follows: if $i,j \leq 2$, the edge is routed exactly as in $Y^{(r)}$ (so its destination lies in $\{1,2\}$); otherwise the edge points to node n. Both tensors agree outside the $\{1,2\}^3$ corner and disagree inside it because $Y^{(a)} \neq Y^{(b)}$, so $X^{(a)} \neq X^{(b)}$.

Sender-axis marginals. We show that $\mathbf{1}^\top X^{(a)}_{:,i,j} = \mathbf{1}^\top X^{(b)}_{:,i,j}$ for every $(i,j) \in [n]^2$, by case analysis on (i,j).

(i) $(i,j) \in \{1,2\}^2$. Any sender $k > 2$ gives $X^{(r)}_{k,i,j} = 0$, since the edge (k,i) has $k > 2$ and therefore points to $n \notin \{1,2\} \ni j$. Hence

$$\mathbf{1}^\top X^{(r)}_{:,i,j} \;=\; \sum_{k=1}^{n} X^{(r)}_{k,i,j} \;=\; \sum_{k=1}^{2} Y^{(r)}_{k,i,j} \;=\; \mathbf{1}^\top Y^{(r)}_{:,i,j} \;=\; 1$$

by the $n = 2$ identity (16) applied to $Y^{(r)}$; in particular this is independent of r.

(ii) $i \le 2$, $j = n$. For $k \le 2$ the edge (k,i) has $k, i \le 2$, so it points into $\{1,2\}$ and hence not to $n = j$, contributing 0. For $k > 2$ the edge (k,i) has $k > 2$, so it points to $n = j$ and contributes 1. Therefore $\mathbf{1}^\top X^{(r)}_{:,i,j} = n - 2$, independent of r.

(iii) $i > 2$, $j = n$. For every k the edge (k,i) has $i > 2$, so it points to $n = j$ and contributes 1. Therefore $\mathbf{1}^\top X^{(r)}_{:,i,j} = n$, independent of r.

(iv) all remaining (i,j)*, i.e.* $j \in \{3,\dots,n-1\}$*, or* $i > 2$ *with* $j < n$. Every edge (k,i) either has $k, i \le 2$ (and points into $\{1,2\}$) or has $k > 2$ or $i > 2$ (and points to n); in neither case does its destination equal j. Therefore $\mathbf{1}^\top X^{(r)}_{:,i,j} = 0$, independent of r.

Receiver-axis marginals. The analogous case analysis with sender and receiver roles swapped (replace each occurrence of "edge (k,i)" by "edge (i,k)") shows $\mathbf{1}^\top X^{(a)}_{i,:,j} = \mathbf{1}^\top X^{(b)}_{i,:,j}$ for every $(i,j) \in [n]^2$. The only case in which the $n = 2$ result is invoked is again $(i,j) \in \{1,2\}^2$: there, $\mathbf{1}^\top X^{(r)}_{i,:,j} = \mathbf{1}^\top Y^{(r)}_{i,:,j} = 1$ by (16). The remaining cases yield $n-2$, n, or 0 exactly as in the sender-axis analysis.

Encoder collision. Substituting these matched marginals into the same closed form derived in the previous subsection,

$$\mathbf{A}^{(r)}_{ij} = \frac{1}{n}\big(\mathbf{1}^\top X^{(r)}_{:,i,j}\big)\,\mathbf{w}^{(1)} + \mathbf{b}^{(1)} \qquad \mathbf{B}^{(r)}_{ij} = \frac{1}{n}\big(\mathbf{1}^\top X^{(r)}_{i,:,j}\big)\,\mathbf{w}^{(2)} + \mathbf{b}^{(2)}$$

gives $\mathbf{A}^{(a)}_{ij} = \mathbf{A}^{(b)}_{ij}$ and $\mathbf{B}^{(a)}_{ij} = \mathbf{B}^{(b)}_{ij}$ for every $(i,j) \in [n]^2$. Hence the pointer-only edge-feature block

$$\mathbf{u}_{X^{(a)}} = \mathbf{A}^{(a)} + \mathbf{B}^{(a)} = \mathbf{A}^{(b)} + \mathbf{B}^{(b)} = \mathbf{u}_{X^{(b)}}$$

while $X^{(a)} \neq X^{(b)}$.