Richer Representations for Neural Algorithmic Reasoning via Auxiliary Reconstruction

Jiafu Huang[1], Chao Peng[1,*], Chenyang Xu[1,*], Zhengfeng Yang[1,*], Kecheng Cai[1], Chenhao Zhang[1], Yi Wang[1], Yiwei Gong[1], Wanqin Zhou[2], Irene Zheng[3]

[1] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
[2] Overseas Education College, Xiamen University, China
[3] Dept. of Electrical and Electronic Engineering, The University of Melbourne, Australia

Abstract

Neural algorithmic reasoning has recently emerged as a popular research direction. It aims to train neural networks to mimic the step-by-step behavior of classical rule-based algorithms. More specifically, the execution of such algorithms can be abstracted as a sequence of states, where each state represents the intermediate outcome after an execution step. The training objective is to generate state sequences that replicate the underlying algorithmic process. A common framework for this task adopts an "encoder-processor-decoder" architecture, where the encoder learns representations of states, the processor simulates algorithmic steps, and the decoder reconstructs output states. While prior work has primarily focused on improving the processor, the role of the encoder in representation learning has received little attention. Most existing methods rely on simple MLP encoders, raising the question of whether such representations are sufficiently informative for supporting algorithmic reasoning.

This paper investigates how to improve encoder representations for neural algorithmic reasoning. We propose a reconstruction module that aims to recover the input state from its encoded representation. This auxiliary reconstruction task encourages the encoder to retain critical information about the input. We demonstrate that incorporating this task during training improves the performance of existing neural architectures on standard benchmarks. Furthermore, we observe that current encoders often underutilize the correlations among features within a state. To address this, we draw inspiration from self-supervised learning and design an enhanced variant of the auxiliary task that encourages the encoder to capture intra-state feature dependencies. Experimental results show that our method enables the encoder to learn richer representations, thereby enhancing the performance of existing processors on algorithmic reasoning tasks.

1 Introduction

Neural algorithmic reasoning (NAR), which bridges the gap between discrete algorithmic logic and continuous neural representations, has received growing attention in recent years. The central goal of this direction is to enable neural networks to perform algorithmic reasoning. In particular, the execution of a classical rule-based algorithm can be abstracted as a sequence of states (see Section 2.1 for more details), and NAR seeks to train neural models to generate similar state sequences given an initial input, thereby mimicking the step-by-step behavior of classical algorithms. This ability to internalize algorithmic structure allows neural networks to serve as algorithmically informed components within larger architectures, supporting tasks such as algorithm-guided planning (Deac et al. 2021), neural program synthesis (Numeroso, Bacciu, and Velickovic 2023), and generalizable reasoning over structured inputs such as graphs and sequences (Velickovic et al. 2022b).

This line of research was first introduced in (Velickovic et al. 2022a), where the authors proposed the now-standard *encoder–processor–decoder* paradigm for solving NAR tasks. In this framework, given an algorithmic input, the encoder transforms the current state into latent representations, the processor then simulates the algorithm's transition dynamics over time, and the decoder maps the processed features into predictions of the next state. This modular architecture enables flexible and scalable learning across a wide range of combinatorial tasks.

Building on this paradigm, many follow-up studies have proposed new network architectures to enhance the model's reasoning capabilities. Examples include Triplet-GMPNN (Ibarz et al. 2022), RNAR (Xu and Velickovic 2024), and G-ForgetNet (Bohde et al. 2024), which each introduce structural innovations to improve performance on NAR tasks. However, we observe that most of these efforts focus on improving the processor component, while the encoder has received comparatively little attention. In most existing methods, the encoder is implemented as simple linear layers, with limited consideration of the structural properties inherent to NAR tasks. This raises the following question:

Can we design NAR-specific encoder architectures that yield richer representations?

Another limitation of the current paradigm lies in its training objective, which relies solely on the final prediction loss. Consequently, it remains unclear whether suboptimal reasoning performance stems from an underperforming processor or an encoder that fails to generate sufficiently expressive representations. This leads to another key question:

*Correspondence to: Chao Peng, Chenyang Xu, Zhengfeng Yang ({cpeng, cyxu, zfyang}@sei.ecnu.edu.cn).

Can we develop a more targeted training objective that guides the encoder independently of the processor?

1.1 Our Contributions

The two questions raised above motivate our work. We address them by proposing a training objective specifically tailored for the encoder, along with a NAR-specific encoder architecture designed to capture the structural properties of algorithmic states, thereby enriching the learned representations and enhancing the model's reasoning capability.

To evaluate the effectiveness of our approach, we conduct experiments on the CLRS benchmark (Velickovic et al. 2022a), a widely adopted benchmark for NAR that presents two key challenges: (1) **Diversity**: The benchmark comprises 30 diverse algorithmic tasks drawn from the classical textbook *Introduction to Algorithms* (Cormen et al. 2009), covering topics such as sorting, searching, dynamic programming, graph traversal, string processing, and more. This diversity makes it highly challenging to design a single architecture that generalizes well across tasks with fundamentally different computational structures. (2) **Generalization**: A key design feature of CLRS is the significant scale gap between the training and test instances. While models are trained on small problem instances, they are evaluated on substantially larger ones, making generalization to unseen scales a critical challenge. Across most tasks, the training set instances contain approximately 10 input elements on average (e.g., list items or graph nodes), whereas the test instances contain up to 64 elements. This creates a substantial difficulty, as models must generalize from small training inputs to much larger and more complex test cases.

Our contributions are summarized as follows:

- We augment the standard encoder–processor–decoder framework with an *additional reconstruction module* and propose an *auxiliary reconstruction objective* that provides direct supervision to the encoder. This objective encourages the encoder to retain essential information about the current algorithmic state, independent of the downstream processor.
- Built on the extended framework, we design a more expressive encoder architecture tailored to NAR tasks. In contrast to simple linear projections, our encoder incorporates a graph neural network and gated residual connections to better capture the algorithmic structures. This design enables the encoder to effectively support both the prediction and reconstruction tasks simultaneously.
- We further introduce a feature-level masking strategy that enhances representation learning within the proposed framework. This strategy is inspired by self-supervised learning and explicitly leverages the correlations between different components of a NAR state, improving the encoder's ability to capture intra-state dependencies.
- We conduct comprehensive experiments on the challenging CLRS benchmark, demonstrating that our method yields consistent improvements across a wide range of algorithmic tasks. ReNAR significantly boosts the average accuracy over baseline models, improving it from 83.63% to 87.11%. M-ReNAR achieves further gains by leveraging the proposed masking strategy, reaching an average accuracy of 88.41%.

1.2 Other Related Works

Representation Learning. Representation learning aims to discover meaningful features from raw data, facilitating downstream tasks without the need for manual feature engineering. It has been a central focus across a wide range of domains, including computer vision (He et al. 2016; Dosovitskiy et al. 2021), natural language processing (Peters et al. 2018; Devlin et al. 2019), and graph-structured data (Kipf and Welling 2017; Hu et al. 2020). A variety of techniques based on self-supervised learning have been developed for effective representation learning. These methods typically rely on auxiliary objectives (Doersch, Gupta, and Efros 2015; Rong et al. 2020) or structural priors (Battaglia et al. 2018) to encourage models to capture semantic or structural patterns inherent in the data.

Self-Supervised Learning. Self-supervised learning has emerged as a powerful paradigm for learning meaningful representations without the need for large-scale labeled data. Techniques such as contrastive learning (Chen et al. 2020; He et al. 2020) and masked prediction(Devlin et al. 2019; Bao et al. 2022) have achieved remarkable success across domains by encouraging models to capture internal correlations and latent semantics. Foundational efforts in graph self-supervised learning have extended these principles to structured data, proposing general objectives such as subgraph-level discrimination and attribute masking (You et al. 2020; Hu et al. 2020), which have proven effective for learning transferable representations. Building on these advances, graph autoencoders, such as SimGRACE(Xia et al. 2022) and GraphMAE(Hou et al. 2022) has emerged, leveraging contrastive or masked reconstruction strategies to further enhance the quality of graph representations.

2 Preliminaries

In this section, we formally introduce the setting of neural algorithmic reasoning and describe the encoder-processor-decoder paradigm commonly used in the literature.

2.1 Classical Algorithm Execution as State Sequences

We begin by discussing how classical rule-based algorithms can be abstracted as sequences of structured states. These algorithms typically maintain internal variables that evolve over time, and the values of these variables at each step can be interpreted as discrete execution states.

To illustrate this idea, consider the depth-first search (DFS) algorithm (Cormen et al. 2009). At each step, DFS maintains a state composed of node-level information[1] including *color*, *discovery time* d, and *finishing time* f. The

[1]We remark that DFS also maintains certain edge-level variables, such as whether an edge has been explored. However, for simplicity, we focus here solely on node-level information to provide a clean illustration.

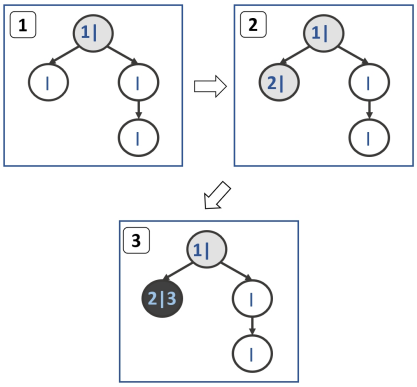

Figure 1: An illustration of three execution steps of the DFS algorithm on a sample graph. Each node is annotated with its discovery and finishing times in the format $d \mid f$, and the color indicates its visitation status.

color of a node indicates its visitation status in the traversal: white for unvisited, gray for discovered but not fully explored, and black for fully explored. The discovery time d records the time when a node is first encountered, and the finishing time f marks the time when the exploration of all its descendants is complete.

Figure 1 illustrates three execution steps of DFS on a given instance. Each state can be viewed as a structured snapshot of the graph, annotated with the corresponding node-level variables. In NAR benchmarks, these variables are typically referred to as *hints* (Velickovic et al. 2022a).

The collection of all (node, hint) pairs at a specific time step constitutes the algorithm's state at that point. The sequence of such states over time forms the complete execution trajectory of the algorithm on the input instance.

2.2 NAR Datasets

A typical NAR dataset is defined with respect to a specific algorithmic problem and the classical algorithm chosen to solve it. Each instance in the dataset consists of a problem input (usually represented as a graph) along with its corresponding execution trace, encoded as a sequence of states.

Formally, let $\mathbf{x}$ denote a problem instance (e.g., a graph with node and edge features), and let $\mathbf{y} = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(T)}\}$ denote the corresponding execution trajectory. Each $\mathbf{y}^{(t)}$ represents the state of the algorithm at time step t, consisting of a set of hints defined over the nodes and edges of the graph. This sequence serves as the target output for neural networks trained to emulate the algorithm in a step-by-step manner.

2.3 Encoder-Processor-Decoder Paradigm

The *encoder–processor–decoder* paradigm serves as the standard framework in most prior work. At each reasoning step, the model receives as input the pair $\left(\mathbf{x}, \mathbf{y}^{(t)}\right)$, where $\mathbf{x}$ is the problem instance and $\mathbf{y}^{(t)}$ is the current state. The goal is to predict the next state $\mathbf{y}^{(t+1)}$.

The encoder module consists of multiple encoder networks. Suppose there are k distinct hint types; accordingly, the encoder module includes k separate encoder networks, each responsible for processing one specific hint. Let $G = (V, E)$ denote the input graph structure. The state at time step t is represented as $\mathbf{y}^{(t)} = \{\mathbf{h}_1^{(t)}, \ldots, \mathbf{h}_k^{(t)}\}$, where each $\mathbf{h}_i^{(t)} = \{h_i^{(t)}(v)\}_{v \in V}$ denotes the value of the i-th hint over all nodes in the graph[2].

Each encoder network takes as input the problem instance $\mathbf{x}$ and a specific hint $\mathbf{h}_i^{(t)}$, and produces a corresponding representation $\mathbf{z}_i^{(t)} = \{z_i^{(t)}(v)\}_{v \in V}$ for all nodes. These hint-wise representations are then aggregated and passed to the processor module, which is typically implemented using a graph neural network. The processor simulates the algorithm's transition dynamics and generates latent node features. These features are then fed into the decoder module, which also consists of k decoder networks. Each decoder is responsible for predicting one specific hint, and together they produce the predicted output state $\hat{\mathbf{y}}^{(t+1)} = \{\hat{\mathbf{h}}_i^{(t+1)}\}_{i \in [k]}$. To supervise the model's predictions, we compare the predicted state $\hat{\mathbf{y}}^{(t+1)}$ with the ground-truth state $\mathbf{y}^{(t+1)}$ at each time step and compute a prediction loss over all hint types. Let $\mathcal{L}_{\text{pred}}^{(t)}$ denote the prediction loss at step t, defined as $\mathcal{L}_{\text{pred}}^{(t)} = \sum_{i=1}^{k} \ell_{\text{hint}}\left(\hat{\mathbf{h}}_i^{(t+1)}, \mathbf{h}_i^{(t+1)}\right)$, where $\hat{\mathbf{h}}_i^{(t+1)}$ and $\mathbf{h}_i^{(t+1)}$ denote the predicted and ground-truth values of the i-th hint, respectively. The function ℓ_{hint} is a hint-specific loss function, chosen based on the type of hint (e.g., cross-entropy for classification-type hints like node color in DFS; mean squared error for regression-type hints like discovery or finishing times). The model is trained by minimizing the total prediction loss across all time steps and training examples. All components of the encoder–processor–decoder architecture are optimized jointly using backpropagation.

3 Incorporating Auxiliary Reconstruction

As described in the previous section, standard neural algorithmic reasoning models are trained to minimize a prediction loss between the predicted state and the ground-truth state. While this loss effectively guides the overall training, it entangles the contributions of the encoder, processor, and decoder, making it difficult to isolate the quality of the encoder's representations. In particular, when the model underperforms, it remains unclear whether the issue lies in the processor's transition modeling or in the encoder's ability to produce informative representations.

To address this ambiguity and to explicitly encourage high-quality representation learning at the encoder stage, we introduce an auxiliary reconstruction module. Reconstruction is a widely used self-supervised objective in representation learning (Vincent et al. 2008; Kingma and Welling

[2]For simplicity, we describe only node-level hints in this illustration and in the descriptions of the following sections. The benchmark datasets may also include edge-level and graph-level hints, which can be handled in a similar manner.

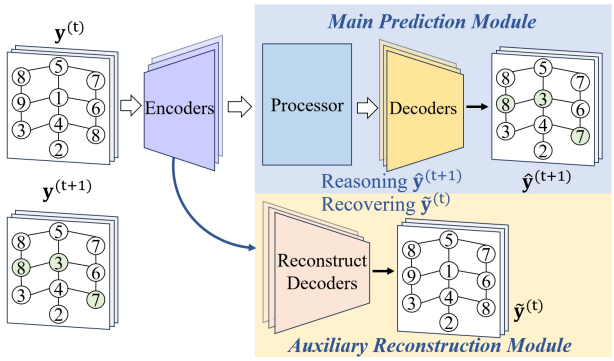

Figure 2: An extended encoder–processor–decoder framework incorporating an auxiliary reconstruction module.

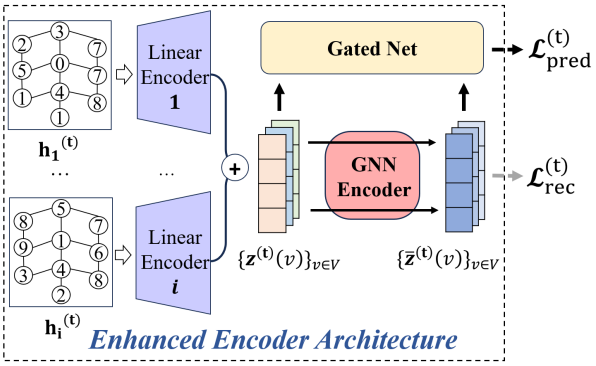

Figure 3: An illustration of the enhanced encoder.

2014; Kipf and Welling 2016), where the goal is to recover the input from its latent representation. By applying this principle, we design an additional loss that directly supervises the encoder output, encouraging it to retain sufficient information about the input state. This auxiliary task complements the main prediction loss and provides a more targeted signal for training the encoder effectively.

3.1 Framework with Reconstruction Module

We now describe the extended framework that incorporates an auxiliary reconstruction module, which provides additional supervision to the encoder by encouraging it to retain information sufficient to recover the input state.

An overview of the framework is shown in Figure 2. The reconstruction module branches from the encoder and operates in parallel with the main prediction pathway. In addition to predicting the next algorithmic state, the encoder outputs are also used to reconstruct the current input state, thereby providing an additional learning signal.

More precisely, the reconstruction module consists of k reconstruction decoders, with the same structure and number as the original decoders. Each reconstruction decoder is responsible for reconstructing one specific hint, similar to how each original decoder predicts a hint in the next state. However, unlike the original decoders, which take the processor's latent output to predict the next state $\hat{\mathbf{y}}^{(t+1)} = \{\hat{\mathbf{h}}_i^{(t+1)}\}_{i\in[k]}$, the reconstruction decoders take as input the encoder representations $\mathbf{z}^{(t)}$ and attempt to reconstruct the current state. That is, they produce $\tilde{\mathbf{y}}^{(t)} = \{\tilde{\mathbf{h}}_i^{(t)}\}_{i\in[k]}$, where each $\tilde{\mathbf{h}}_i^{(t)}$ is the reconstruction of the hint $\mathbf{h}_i^{(t)}$ from input state $\mathbf{y}^{(t)}$ at time step t.

To supervise the reconstruction branch, we define a reconstruction loss that compares the reconstructed state $\tilde{\mathbf{y}}^{(t)}$ with the input state $\mathbf{y}^{(t)}$. Similar to the prediction loss, the reconstruction loss is computed over all hint types $\mathcal{L}_{\text{rec}}^{(t)} = \sum_{i=1}^{k} \ell_{\text{hint}}\left(\tilde{\mathbf{h}}_i^{(t)}, \mathbf{h}_i^{(t)}\right)$, where the same hint-specific loss function ℓ_{hint} used in the prediction loss is applied here.

To jointly optimize both the prediction and reconstruction objectives, we combine the two losses at each time step with a weighting coefficient λ. The total loss is defined as $\mathcal{L}_{\text{total}} = \sum_{t=1}^{T-1}\left(\mathcal{L}_{\text{pred}}^{(t)} + \lambda \cdot \mathcal{L}_{\text{rec}}^{(t)}\right)$.

This training objective supervises both the main prediction path and the auxiliary reconstruction path, encouraging the encoder to produce representations that are not only useful for predicting future states but also sufficiently informative to recover the current state.

3.2 Reconstruction-Oriented Encoder Design

In the framework introduced above, the auxiliary reconstruction module places additional demands on the encoder: it must not only support accurate prediction of future states through the processor but also preserve sufficient information to reconstruct the current state directly. To meet this dual objective, we revisit the design of the encoder networks.

In previous NAR methods, each encoder network is typically implemented as a simple linear layer applied to the corresponding hint. However, this design[3] remains limited in its ability to capture structural context. To further enhance the encoder's expressive power, we augment encoder networks with a graph neural network layer and a gated network. These modifications are designed to strengthen the encoder's capacity to model local structure and retain key information relevant for reconstruction.

An illustration of the new encoder module is shown in Figure 3. Each hint $\mathbf{h}_i^{(t)}$ is first passed through its corresponding linear layer, producing node-level latent features $\{z_i^{(t)}(v)\}_{v\in V}$ for the i-th hint. These features are then aggregated across hints for each node to form the initial embedding for the GNN encoder: $\mathbf{z}^{(t)}(v) = \sum_{i\in[k]} z_i^{(t)}(v)$. The GNN then computes updated node representations $\bar{\mathbf{z}}^{(t)}(v)$, which are used as input to the reconstruction decoders.

For the processor, we further refine the representation by combining $\bar{\mathbf{z}}^{(t)}(v)$ with the initial embedding $\mathbf{z}^{(t)}(v)$ through a gated mechanism:

$$\mathbf{z}^{(t)}(v) = g_v \odot \bar{\mathbf{z}}^{(t)}(v) + (1 - g_v) \odot \mathbf{z}^{(t)}(v),$$

where $g_v \in [0, 1]$ is a learned gating coefficient for node v, and $\odot$ denotes element-wise multiplication.

[3] Additional empirical evidence is provided in the full version of this paper.

Intuitively, incorporating a GNN into the encoder allows each node to aggregate local structural information at an earlier stage, enabling the encoder to generate more context-aware representations before interacting with the processor. Although the processor itself is also typically implemented as a GNN, its role is to simulate the algorithm's transition dynamics over time. In contrast, the encoder's role is to extract rich, informative features from the current state alone. The addition of a gated network helps preserve the original hint-specific signals, such as the raw input values for each node, while also incorporating contextual information aggregated from neighboring nodes. These enhancements help the encoder provide a stronger input to the processor, ultimately leading to better performance.

4 Advanced Reconstruction via Hint-Level Masking

The reconstruction framework introduced in the previous section supervises the new encoder module, which is shown in Figure 3, by requiring it to reconstruct all node-level hints in the current state. While this approach encourages the new encoder to preserve essential information, it, like previous encoder designs, treats each hint independently and does not explicitly model their interactions. In practice, however, many algorithmic states exhibit strong internal correlations across hints. Exploiting these correlations can provide additional inductive bias to help the encoder learn more structured and generalizable representations.

4.1 Hint Dependencies

Algorithmic reasoning tasks often involve multiple node-level hints that are tightly coupled due to the underlying algorithmic semantics. Capturing these correlations is important for learning coherent representations.

For example, consider the DFS algorithm shown in Figure 1, where each node is annotated with three hints: color, discovery time, and finish time. These hints are logically interdependent:

- If a node has neither discovery nor finish time, its color must be white;
- If the discovery time is present but the finish time is not, the node must be gray;
- If both timestamps are available, the color must be black.

These relationships are not imposed by external labels, but are an intrinsic part of the algorithm's execution dynamics. Such inter-hint dependencies are also common in other algorithms, such as shortest path (where distance and predecessor are correlated). However, existing NAR frameworks typically process each hint independently in the encoder, missing the opportunity to exploit these structural regularities.

4.2 Hint-Level Masking Strategy

To encourage the new encoder to model cross-hint dependencies, we adopt a hint-level masking strategy inspired by recent advances in self-supervised learning (Hou et al. 2023; Li et al. 2023; Wang et al. 2024). Rather than treating all node-hint pairs equally during reconstruction, we selectively

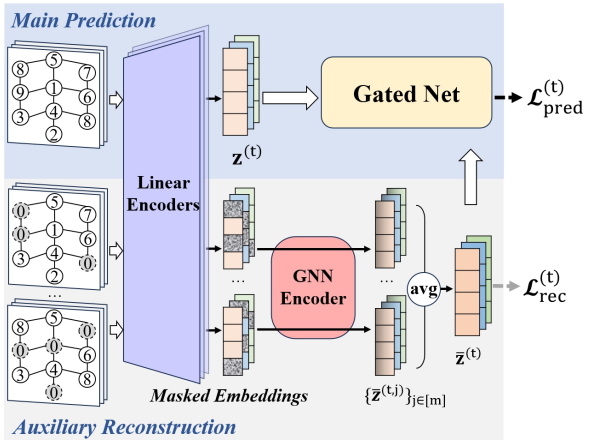

Figure 4: An illustration of the hint-level masking strategy.

mask a subset of the input hints and train the encoder to recover only the masked values. This forces the encoder to infer missing information based on the context provided by the remaining unmasked hints.

Formally, we fix a masking ratio $\beta \in (0, 1)$, which specifies the proportion of node-hint pairs to be masked in each reconstruction instance. Let $m = \lfloor 1/\beta \rfloor$ denote the number of independent masking rounds. For each round $j \in [m]$, we independently sample a set of masked indices $\mathcal{M}^{(j)} \subseteq V \times [k]$, where $|\mathcal{M}^{(j)}| = \lfloor \beta \cdot k \cdot |V| \rfloor$. For each $(v, i) \in \mathcal{M}^{(j)}$, the corresponding input hint $h_i^{(t)}(v)$ is masked by setting its value to zero. The resulting masked input at round j, denoted by $\mathbf{h}^{(t,j)}$, is defined element-wise as:

$$h_i^{(t,j)}(v) = \begin{cases} 0, & \text{if } (v,i) \in \mathcal{M}^{(j)} \\ h_i^{(t)}(v), & \text{otherwise.} \end{cases}$$

The linear layers and GNN within the new encoder module, then process each masked version of the input independently, producing a collection of hidden representations $\{\bar{\mathbf{z}}^{(t,j)}\}_{j\in[m]}$. These representations are aggregated and averaged to produce a unified representation: $\bar{\mathbf{z}}^{(t)} = \frac{1}{m}\sum_{j=1}^{m} \bar{\mathbf{z}}^{(t,j)}$. Subsequently, the unified representation $\bar{\mathbf{z}}^{(t)}$ is passed to the gated network along with $\mathbf{z}^{(t)}$, which is obtained by encoding the unmasked data through linear layers, to support the main prediction.

In parallel, each reconstruction decoder receives its corresponding masked input $\bar{\mathbf{z}}^{(t,j)}$ and attempts to reconstruct only the masked hints $(v, i) \in \mathcal{M}^{(j)}$. The new reconstruction loss is defined as the sum over the masked positions: $\mathcal{L}^{(t)}_{\text{rec}} = \sum_{j=1}^{m} \sum_{(v,i)\in\mathcal{M}^{(j)}} \ell_{\text{hint}}\left(\tilde{h}_i^{(t,j)}(v), h_i^{(t)}(v)\right)$, where $\tilde{h}_i^{(t,j)}(v)$ denotes the reconstruction decoder's prediction for the node-hint pair (v, i) in the j-th masking round.

An illustration of this masking and reconstruction process is provided in Figure 4. This masking strategy enables the encoder to learn more structured representations by leveraging dependencies between hints. It also introduces stochasticity into training, which we find improves generalization in downstream reasoning tasks.

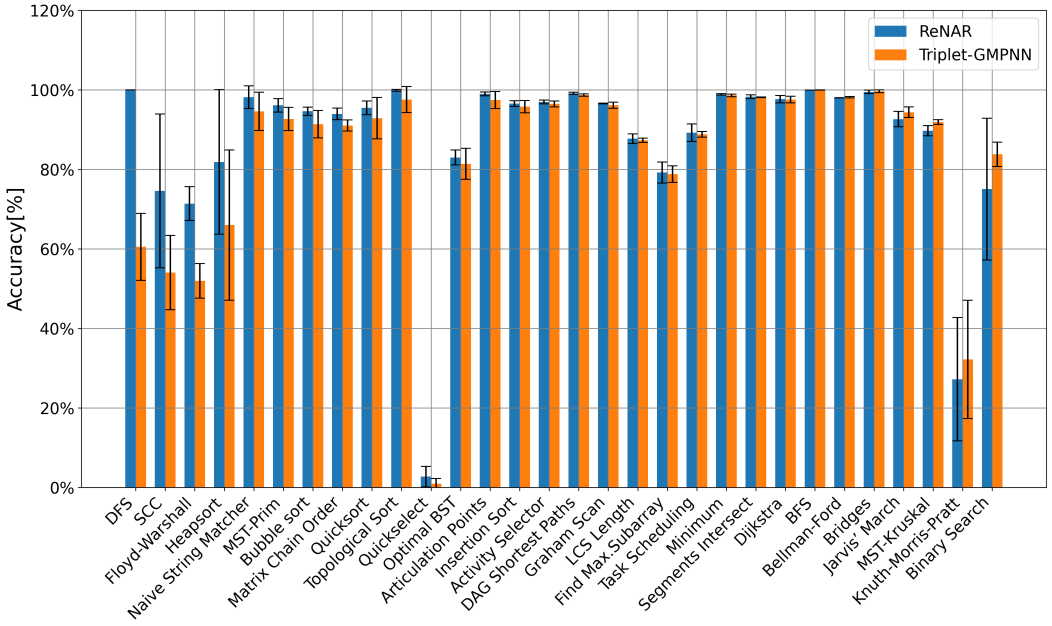

Figure 5: Comparison of the Triplet-GMPNN (Ibarz et al. 2022) architecture's performance before and after augmentation with ReNAR. The 30 algorithmic tasks are arranged in descending order of the accuracy.

Task Category	Prior Best	CEF-GMPNN	G-ForgetNet	Triplet-GMPNN	ReNAR	M-ReNAR
Graphs	64.98% ±2.59	86.99% ±1.78	88.80% ±0.84	86.68% ±2.73	93.75% ±2.40	**94.74% ±0.73**
Geometry	92.48% ±1.35	95.38% ±1.11	95.09% ±1.16	**96.21% ±0.74**	95.83% ±0.82	95.83% ±0.41
Strings	4.08% ±0.57	70.86% ±6.47	54.74% ±1.95	63.40% ±9.85	62.70% ±9.17	**74.72% ±18.72**
DP	76.00% ±2.47	86.23% ±1.77	86.70% ±0.49	86.59% ±1.95	**88.23% ±1.52**	87.55% ±1.76
Div. & C.	65.23% ±2.56	76.38% ±3.04	78.97% ±0.70	78.81% ±2.10	79.22% ±2.67	**81.54% ±2.71**
Greedy	84.13% ±2.59	92.61% ±0.44	91.79% ±0.23	92.65% ±0.73	93.10% ±1.34	**94.35% ±0.46**
Search	56.11% ±0.36	61.74% ±0.45	**63.84% ±0.84**	61.14% ±1.57	58.89% ±6.86	61.22% ±2.04
Sorting	71.53% ±0.97	81.52% ±3.18	78.09% ±3.78	86.52% ±7.28	**92.13% ±5.41**	90.51% ±3.77
Overall Average	66.04%	82.68%	82.89%	83.63%	87.11%	**88.41%**

Table 1: Summary of our results across each task category in the CLRS benchmark. The best-performing results are highlighted in bold, and the second-best results are underlined.

5 Experiments

This section evaluates the effectiveness of our proposed methods on the challenging CLRS benchmark, which spans all 30 algorithmic tasks across 8 major categories. Our experiments are designed to address the following questions:

- Can our framework, along with the proposed encoder architecture, be effectively integrated with existing processor modules to enhance reasoning capability?
- Does the hint-level masking strategy further improve performance, and how does the choice of masking ratio affect performance across different tasks?

We refer to our reconstruction-augmented framework as ReNAR, and the masking-enhanced variant as M-ReNAR. To investigate the first question, we integrate ReNAR with several existing processor architectures to assess whether it consistently improves reasoning performance. Due to space limitations, we report results using the widely adopted Triplet-GMPNN processor (Ibarz et al. 2022) in the main body, and defer the results with other processors to the full version of this paper. To answer the second question, we evaluate the performance of M-ReNAR under varying mask ratios β. We also conduct ablation studies to isolate the contributions of individual components, with the corresponding results also provided in the full version of this paper.

5.1 Setup

Baselines. We compare our methods against several recent approaches, including Triplet-GMPNN (Ibarz et al. 2022), CEF-GMPNN (Shi et al. 2024), and G-ForgetNet (Bohde et al. 2024). We also include earlier baselines such as MPNN (Gilmer et al. 2017), PGN (Velickovic et al. 2020), MemNet (Velickovic et al. 2022a), and NPQ (Jain, Velickovic, and Liò 2023). For these methods, we report the best result across all algorithmic tasks and denote it as the *prior best*. We follow the standard evaluation setting on CLRS and

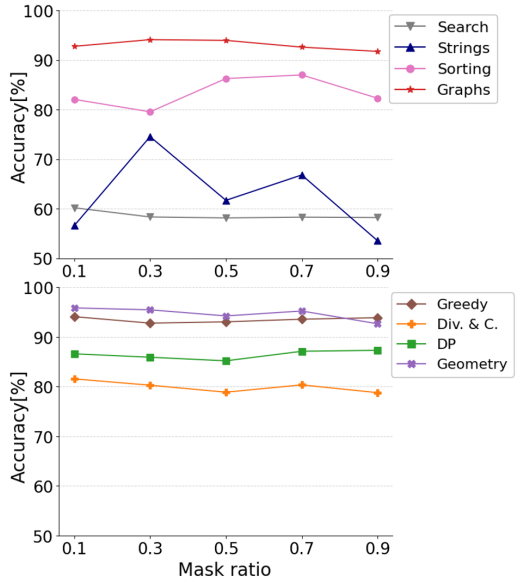

Figure 6: Performance of M-ReNAR under different masking ratios, grouped by algorithmic category.

report the F1 score as the primary metric, which is computed by averaging over all hints at each step.

Computational Details. Our experiments are conducted on multiple machines equipped with different NVIDIA GPUs, including RTX 4090, RTX A100, and RTX A6000. All reported results are averaged over four runs. To ensure fair comparisons, we follow the widely used experimental hyperparameter settings from (Ibarz et al. 2022): the batch size is set to 32, and the models are trained for 10,000 steps using the Adam optimizer with a learning rate of 0.001. For ReNAR, we set the weight of the reconstruction loss to $\lambda = 0.1$. For M-ReNAR, which includes masked reconstruction, we use a higher $\lambda = 1$ to emphasize the supervision signal. The average training time is approximately 0.79 GPU hours for ReNAR and 1.30 GPU hours for M-ReNAR, while the compared baseline method (Ibarz et al. 2022) has a training time of 0.52 GPUs.

5.2 Effectiveness of Auxiliary Reconstruction

This subsection evaluates the effectiveness of the proposed framework, focusing on whether it enhances the encoder's representation quality and improves reasoning performance when integrated with existing processors. We observe that different processors exhibit consistent trends. Therefore, as mentioned earlier, we report only the results based on the Triplet-GMPNN processor in the main body and defer the remaining results to the full version of the paper.

As shown in Figure 5, augmenting existing processors with our reconstruction framework leads to consistent improvements in reasoning performance across most of tasks. The gains are particularly pronounced on graph-based algorithms and sorting-related problems. For example, on the DFS reasoning task, the accuracy improves dramatically, from around 60% to nearly 100%.

5.3 Impact of Hint-Level Masking

This subsection investigates the role of the hint-level masking strategy in further improving performance and analyzes how the masking ratio β affects different algorithmic tasks. In our experiments, M-ReNAR is evaluated under a range of mask ratios $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. We observe that tasks within the same algorithmic category tend to exhibit similar sensitivities to the masking ratio. Therefore, we compute the average accuracy for each algorithm category at different masking ratios. The results are presented in Figure 6.

From the figure, we observe that the sensitivity to the masking ratio varies across algorithmic categories. Different algorithmic categories achieve their best performance at different values of β. Notably, graph-based tasks are relatively stable across all settings, while string-related tasks exhibit more pronounced fluctuations. This may be due to the fact that string algorithms often rely heavily on precise token-level information, making them more sensitive to partial hint removal during training.

We summarize the comparison between our methods and the baselines in Table 1. For the M-ReNAR column, we report the best performance achieved across all tested masking ratios. The table indicates that incorporating the reconstruction framework improves the average accuracy across the entire benchmark from 83.63% to 87.11%. With the addition of the hint-level masking strategy, the performance is further boosted to 88.41%. Our methods demonstrate consistent improvements across most algorithmic categories, with particularly notable gains on graph-based tasks, where the best average accuracy improves from 88.80% to 94.74%.

5.4 Discussion

We see the following trends from the experimental results:

- The proposed ReNAR framework consistently improves reasoning performance. The reconstruction objective provides direct supervision to the encoder, encouraging it to retain more task-relevant information. This, in turn, leads to stronger intermediate features that better support downstream algorithmic reasoning.
- The hint-level masking strategy further strengthens the reconstruction framework by enabling the encoder to capture correlations among different hints. This leads to richer representations and results in additional performance gains on the benchmark.

6 Conclusion

This paper introduces ReNAR, a reconstruction-augmented framework for neural algorithmic reasoning, along with a tailored encoder architecture and a hint-level masking strategy to enhance representation learning. Experiments demonstrate that our method consistently improves reasoning performance across diverse algorithmic tasks, highlighting the importance of direct encoder supervision and intra-state feature modeling. There remain many directions for future work. For example, exploring other forms of auxiliary supervision, like contrastive or predictive objectives, may further enhance representation quality.

Acknowledgements

This work is supported by the National Key Research and Development Program of China (2023YFA1009402, 2025YFC2423000), NSFC Programs (62302166, 62161146001, 62372176), Shanghai Key Lab of Trustworthy Computing, Shanghai Frontiers Science Center of Molecule Intelligent Syntheses and Fundamental Research Funds for the Central Universities.

References

Bao, H.; Dong, L.; Piao, S.; and Wei, F. 2022. BEiT: BERT Pre-Training of Image Transformers. In *ICLR*. OpenReview.net.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V. F.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; Gülçehre, Ç.; Song, H. F.; Ballard, A. J.; Gilmer, J.; Dahl, G. E.; Vaswani, A.; Allen, K. R.; Nash, C.; Langston, V.; Dyer, C.; Heess, N.; Wierstra, D.; Kohli, P.; Botvinick, M. M.; Vinyals, O.; Li, Y.; and Pascanu, R. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261.

Bohde, M.; Liu, M.; Saxton, A.; and Ji, S. 2024. On the Markov Property of Neural Algorithmic Reasoning: Analyses and Methods. In *ICLR*. OpenReview.net.

Chen, T.; Kornblith, S.; Norouzi, M.; and Hinton, G. E. 2020. A Simple Framework for Contrastive Learning of Visual Representations. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, 1597–1607. PMLR.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms, 3rd Edition*. MIT Press.

Deac, A.; Velickovic, P.; Milinkovic, O.; Bacon, P.; Tang, J.; and Nikolic, M. 2021. Neural Algorithmic Reasoners are Implicit Planners. In *NeurIPS*, 15529–15542.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT (1)*, 4171–4186. Association for Computational Linguistics.

Doersch, C.; Gupta, A.; and Efros, A. A. 2015. Unsupervised Visual Representation Learning by Context Prediction. In *ICCV*, 1422–1430. IEEE Computer Society.

Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; Uszkoreit, J.; and Houlsby, N. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*. OpenReview.net.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, 1263–1272. PMLR.

He, K.; Fan, H.; Wu, Y.; Xie, S.; and Girshick, R. B. 2020. Momentum Contrast for Unsupervised Visual Representation Learning. In *CVPR*, 9726–9735. Computer Vision Foundation / IEEE.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *CVPR*, 770–778. IEEE Computer Society.

Hou, Z.; He, Y.; Cen, Y.; Liu, X.; Dong, Y.; Kharlamov, E.; and Tang, J. 2023. GraphMAE2: A Decoding-Enhanced Masked Self-Supervised Graph Learner. In *WWW*, 737–746. ACM.

Hou, Z.; Liu, X.; Cen, Y.; Dong, Y.; Yang, H.; Wang, C.; and Tang, J. 2022. GraphMAE: Self-Supervised Masked Graph Autoencoders. In *KDD*, 594–604. ACM.

Hu, W.; Liu, B.; Gomes, J.; Zitnik, M.; Liang, P.; Pande, V. S.; and Leskovec, J. 2020. Strategies for Pre-training Graph Neural Networks. In *ICLR*. OpenReview.net.

Ibarz, B.; Kurin, V.; Papamakarios, G.; Nikiforou, K.; Bennani, M.; Csordás, R.; Dudzik, A. J.; Bosnjak, M.; Vitvitskyi, A.; Rubanova, Y.; Deac, A.; Bevilacqua, B.; Ganin, Y.; Blundell, C.; and Velickovic, P. 2022. A Generalist Neural Algorithmic Learner. In *LoG*, volume 198 of *Proceedings of Machine Learning Research*, 2. PMLR.

Jain, R.; Velickovic, P.; and Liò, P. 2023. Neural Priority Queues for Graph Neural Networks. In *The 2023 ICML, Workshop on Knowledge and Logical Reasoning in the Era of Data-driven Learning*, volume 202. PMLR.

Kingma, D. P.; and Welling, M. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

Kipf, T. N.; and Welling, M. 2016. Variational Graph Auto-Encoders. *CoRR*, abs/1611.07308.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR (Poster)*. OpenReview.net.

Li, J.; Wu, R.; Sun, W.; Chen, L.; Tian, S.; Zhu, L.; Meng, C.; Zheng, Z.; and Wang, W. 2023. What's Behind the Mask: Understanding Masked Graph Modeling for Graph Autoencoders. In *KDD*, 1268–1279. ACM.

Numeroso, D.; Bacciu, D.; and Velickovic, P. 2023. Dual Algorithmic Reasoning. In *ICLR*. OpenReview.net.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

Rong, Y.; Bian, Y.; Xu, T.; Xie, W.; Wei, Y.; Huang, W.; and Huang, J. 2020. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In *NeurIPS*.

Shi, S.; Peng, C.; Xu, C.; and Yang, Z. 2024. A Context-Enhanced Framework for Sequential Graph Reasoning. In *IJCAI*, 4902–4910. ijcai.org.

Velickovic, P.; Badia, A. P.; Budden, D.; Pascanu, R.; Banino, A.; Dashevskiy, M.; Hadsell, R.; and Blundell, C. 2022a. The CLRS Algorithmic Reasoning Benchmark. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, 22084–22102. PMLR.

Velickovic, P.; Bosnjak, M.; Kipf, T.; Lerchner, A.; Hadsell, R.; Pascanu, R.; and Blundell, C. 2022b. Reasoning-Modulated Representations. In *LoG*, volume 198 of *Proceedings of Machine Learning Research*, 50. PMLR.

Velickovic, P.; Buesing, L.; Overlan, M. C.; Pascanu, R.; Vinyals, O.; and Blundell, C. 2020. Pointer Graph Networks. In *NeurIPS*.

Vincent, P.; Larochelle, H.; Bengio, Y.; and Manzagol, P. 2008. Extracting and composing robust features with denoising autoencoders. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, 1096–1103. ACM.

Wang, Y.; Yan, X.; Hu, C.; Xu, Q.; Yang, C.; Fu, F.; Zhang, W.; Wang, H.; Du, B.; and Jiang, J. 2024. Generative and Contrastive Paradigms Are Complementary for Graph Self-Supervised Learning. In *ICDE*, 3364–3378. IEEE.

Xia, J.; Wu, L.; Chen, J.; Hu, B.; and Li, S. Z. 2022. SimGRACE: A Simple Framework for Graph Contrastive Learning without Data Augmentation. In *WWW*, 1070–1079. ACM.

Xu, K.; and Velickovic, P. 2024. Recurrent Aggregators in Neural Algorithmic Reasoning. *CoRR*, abs/2409.07154.

You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; and Shen, Y. 2020. Graph Contrastive Learning with Augmentations. In *NeurIPS*.