Alignment of MPNNs and Graph Transformers

Bao Nguyen [*1] **Anjana Yodaiken** [*1] **Petar Veličković** [2]

Abstract

As the complexity of machine learning (ML) model architectures increases, it is important to understand to what degree simpler and more efficient architectures can align with their complex counterparts. In this paper, we investigate the degree to which a Message Passing Neural Network (MPNN) can operate similarly to a Graph Transformer. We do this by training an MPNN to align with the intermediate embeddings of a Relational Transformer (RT). Throughout this process, we explore variations of the standard MPNN and assess the impact of different components on the degree of alignment. Our findings suggest that an MPNN can align to RT and the most important components that affect the alignment are the MPNN's permutation invariant aggregation function, virtual node and layer normalisation.

Code available at: `clrs_dataset_generator` [1] and `gnn_transformer_alignment` [2]

1. Introduction

Graph Neural Networks (GNNs) have emerged as a powerful framework for learning representations in domains where data can be naturally constructed as graphs. This includes various applications, from social network analysis (Li et al., 2023) and recommendation systems (Wu et al., 2022) to drug discovery (Han et al., 2021). GNNs excel in these tasks by imposing a relational inductive bias to leverage the relationships embedded in the underlying topology.

Amongst the various GNN architectures, Message Passing Neural Networks (MPNNs) (Gilmer et al., 2017) are simple yet powerful models for graph representation learning, exploiting local connectivity patterns through iterative message-passing mechanisms.

Alongside the development of MPNNs, the Transformer (Vaswani et al., 2017) architecture has set new benchmarks across a range of domains, notably in Natural Language Processing, by leveraging global self-attention mechanisms to capture long-range dependencies within sequential data. To take advantage of the Transformer's ability to capture these dependencies within the context of graph-learning, Dwivedi and Bresson (2021) developed the Graph Transformer. This development adapts the Transformer's global self-attention mechanism to operate on graphs, enabling the model to capture local and global relationships within the data. However, the Graph Transformer introduces increased complexity, difficulty in training and is computationally expensive.

Since the development of the Graph Transformer (Dwivedi and Bresson, 2021), several adaptations and attempts have been made to improve its performance. This includes the Relational Transformer (RT) (Diao and Loynd, 2023), with its primary extension being the inclusion of edges as first-class model components.

As the complexity of these architectures increases, an important research question arises: to what extent can simpler models operate in the same way as their more complex counterparts? Therefore, this project aims to investigate the extent to which the MPNN and its variants can imitate a Graph Transformer (RT). More specifically, we strive to answer the following questions:

1. Can we train an MPNN and its variants through gradient descent to learn the intermediate embeddings of the Transformer?
2. What components of an MPNN are the most important for aligning to the Transformer's embedding space?
3. How do the MPNN variants perform in and out-of-distribution (OOD) in terms of embedding alignment and from a model distillation (Hinton et al., 2015) perspective?

[*]Equal contribution [1]Department of Computer Science and Technology [2]Google DeepMind. Correspondence to: Bao Nguyen <btn21@cam.ac.uk>, Anjana Yodaiken <agmy2@cam.ac.uk>, Petar Veličković <petarv@google.com>.

Proceedings of the Geometry-grounded Representation Learning and Generative Modeling at 41st International Conference on Machine Learning, Vienna, Austria. PMLR Vol Number, 2024.

[1]https://github.com/anjana-yodaiken/clrs_dataset_generator
[2]https://github.com/baon6052/gnn_transformer_alignment

2. Background & Related Works

2.1. Message Passing Neural Networks

Within the context of molecular chemistry, many models existed that incorporated some form of message-passing and could be described by a unifying framework. Through their abstraction of the most promising commonalities from these message-passing models, Gilmer et al. (2017) proposed the MPNN as a general framework for graph-based learning.

$$\mathbf{n}_i^{(l+1)} = \phi\left(\mathbf{n}_i^{(l)}, \bigoplus_{v \in \mathcal{N}_i} \psi\left(\mathbf{n}_i^{(l)}, \mathbf{n}_j^{(l)}\right)\right) \tag{1}$$

Where ϕ is a *message passing function*; $\bigoplus$ is a *permutation-invariant aggregation function*, such as *sum*, *mean* or *max*, which aggregates all messages coming to node i from its neighbourhood $\mathcal{N}_i$. ψ is a *readout function* that computes the message from node j to node i. Commonly, both ϕ and ψ are multi-layer perceptrons.

Since their introduction, there have been various extensions and adaptations to the initial implementation of the MPNN. GraphSAGE (Hamilton et al., 2017) extends the MPNN framework by introducing a sampling strategy for efficiently aggregating messages from a node's neighbours, allowing it to scale to large graphs. The Graph Attention Network (GAT) (Veličković et al., 2018) uses an attention mechanism to weigh messages from a node's neighbours before aggregating them. This allows each node to focus on the most relevant messages during the update phase.

2.2. Relational Transformer

The Relational Transformer (RT) (Diao and Loynd, 2023) extends on the original transformer architecture (Dwivedi and Bresson, 2021) by introducing directed edge vectors as first-class model components. Thus, edges are incorporated into the self-attention mechanism to update the node representations (see Section 2.2.1).

To avoid the $\mathcal{O}(N^3)$ complexity associated with self-attention for edge updates, Diao and Loynd (2023) update edges using *localised edge updates* (see Section 2.2.2). This method enables the model to retain an $\mathcal{O}(N^2)$ complexity while enriching the edge representations with information about its local neighbourhood.

2.2.1. Extension of Relational Attention

For a node $\mathbf{n}_i$, Diao and Loynd (2023) modify the standard equation for transformer attention to condition the Query, Key and Value (QKV) vectors on the node $\mathbf{n}_i$, as well as the directed edge $\mathbf{e}_{ji}$ between nodes $\mathbf{n}_j$ and $\mathbf{n}_i$. Figure 1 depicts this adaptation to standard transformer attention.

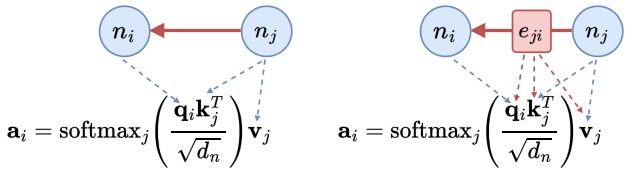

Figure 1: Left to right: 1.) Standard transformer attention: conditions on node vectors only, 2.) Relational attention: conditions on node vectors and the intervening edge embedding. Where d_n is the node vector size, a_i is node i's attention, $\mathbf{q}$ is the query, $\mathbf{k}$ is the key and $\mathbf{v}$ is the value. (Diao and Loynd, 2023)

Equation (2) expresses the conditioning of query ($\mathbf{q}_{ij}$), key ($\mathbf{k}_{ij}$), and value ($\mathbf{v}_{ij}$) vectors on the edge vector, $\mathbf{e}_{ji}$, by concatenating them with the corresponding node vector, $\mathbf{n}_j$, before the linear transformation. Diao and Loynd (2023) reformulate Equation (2) accurately and efficiently in Equation (3) by decomposing the learned weight matrices, $\mathbf{W}_Q$, $\mathbf{W}_K$ and $\mathbf{W}_V$, into two separate matrices for the edge, $\mathbf{W}_e$, and the node, $\mathbf{W}_n$, respectively.

$$\begin{aligned} \mathbf{q}_{ij} &= \mathbf{W}^Q[\mathbf{n}_i, \mathbf{e}_{ji}], \\ \mathbf{k}_{ij} &= \mathbf{W}^K[\mathbf{n}_j, \mathbf{e}_{ji}], \\ \mathbf{v}_{ij} &= \mathbf{W}^V[\mathbf{n}_j, \mathbf{e}_{ji}] \end{aligned} \tag{2}$$

$$\begin{aligned} \mathbf{q}_{ij} &= (\mathbf{W}_n^Q\mathbf{n}_i + \mathbf{W}_e^Q\mathbf{e}_{ji}), \\ \mathbf{k}_{ij} &= (\mathbf{W}_n^K\mathbf{n}_j + \mathbf{W}_e^K\mathbf{e}_{ji}), \\ \mathbf{v}_{ij} &= (\mathbf{W}_n^V\mathbf{n}_j + \mathbf{W}_e^V\mathbf{e}_{ji}) \end{aligned} \tag{3}$$

2.2.2. Localised Edge Updates

RT introduces a mechanism, called *localised edge updates*, to update edge vectors ($\mathbf{e}_{ji}$) in each layer, improving the expressivity of edge embeddings, such that they better describe the relationship between nodes.

The edge updates restrict the edge's aggregation function to consist of its two adjoining nodes, the edge itself and the directed edge running in the opposite direction. Updating edges in this way allows RT to retain its $\mathcal{O}(N^2)$ complexity while enriching edge representations. This method avoids edge self-attention's $\mathcal{O}(N^3)$ complexity.

Edges are updated as follows:

$$\mathbf{e}_{ji}^{(l+1)} = \phi_e\left(\mathbf{e}_{ij}^{(l)}, \mathbf{e}_{ji}^{(l)}, \mathbf{n}_i^{(l+1)}, \mathbf{n}_j^{(l+1)}\right) \tag{4}$$

where l denotes the attention layer, ϕ_e is a function that updates the edge vector based on its inputs (see Appendix A).

Remark. Localised edge updates can be seen as a form of restricted *Triplet Edge Processing* (Dudzik and Veličković,

2022). Triplet Edge Processing uses three connected nodes and the corresponding edges to update the *node* representation (Ibarz et al., 2022). Whereas RT's localised edge updates use the two connected nodes and the corresponding edges to update the *edge* representation.

2.3. MPNN and Transformer Alignment

RTs and MPNNs diverge in their approach to handling relational data. This is primarily due to their structural differences. RTs apply fully connected self-attention and localised edge updates, maintaining $\mathcal{O}(N^2)$ computational complexity. On the other hand, MPNNs operating on sparsely connected graphs typically exhibit lower computational complexity. However, this complexity is contingent on the graph's density. RTs account for long-range dependencies in a single step. In contrast, MPNNs utilise multiple message-passing iterations to handle these long-range connections effectively. In representation learning, both MPNNs and RTs strive to distil meaningful and task-relevant information from data. Where RTs leverage self-attention to craft context-rich embeddings, MPNNs aggregate features from local neighbourhoods, possibly over several layers, to capture broader relational information.

2.3.1. Virtual Node

Cai et al. (2023), in their paper *On the Connection Between MPNN and Graph Transformer*, investigate the alignment of MPNNs and Graph Transformers. They show that MPNNs, when augmented with a virtual node (VN) (Gilmer et al., 2017), can approximate the self-attention mechanism of Graph Transformers under certain conditions. Specifically, they illustrate that to approximate a self-attention layer in Performer/Linear Transformer models, an MPNN + VN configuration requires only $\mathcal{O}(1)$ depth and $\mathcal{O}(1)$ width. For a full approximation of the self-attention mechanism, MPNN + VN can achieve this with $\mathcal{O}(N)$ depth while still maintaining $\mathcal{O}(1)$ width, under specific assumptions about the input graph's features.

Rosenbluth et al. (2024) compared Graph Transformers to MPNNs with VNs. The use of self-attention vs VNs in their global computation is the primary difference between the two architectures. They found that in the non-uniform setting the models with positional encoding implied universality.

The work of Cai et al. (2023) and Rosenbluth et al. (2024) motivate our use of VNs as an approximation of the Graph Transformer's self-attention mechanism to improve in the alignment between the two architectures.

2.3.2. Attention

Before attempting to empirically align the embeddings of the MPNN and RT, it is important to understand to what extent the two architectures can align theoretically. To theoretically assess the alignment between the Transformer and MPNN architectures, it is helpful to conceptualise the Transformer's attention mechanism in terms that allow for its incorporation into the MPNN equation. A requirement of the MPNN's message-passing operation is to observe the properties of a *commutative monoid* [3]. Veličković (2023) shows that a Transformer's attention mechanism can be reformulated as a commutative monoid and integrated with the MPNNs message-passing operation. We can formulate the Transformer attention as:

$$\mathbf{y}_i = \sum_{j \in \mathcal{N}_i} \alpha_j \mathbf{v}_j = \frac{\sum_{j \in \mathcal{N}_i} \left(\exp(\mathbf{q}_i^T \mathbf{k}_j)\right) \mathbf{v}_j}{\sum_{j \in \mathcal{N}_i} \exp(\mathbf{q}_i^T \mathbf{k}_j)} \tag{5}$$

where for a node, $\mathbf{n}_i$, $\mathbf{v}_j = \mathbf{V}\mathbf{n}_j$ is the value associated with a neighbouring node, $\mathbf{n}_j$, and $\alpha_j \in \mathbb{R}$ is the attention value resulting from the softmax-normalised dot product of the query $\mathbf{q}_i = \mathbf{Q}\mathbf{n}_i$ and key $\mathbf{k}_j = \mathbf{K}\mathbf{n}_j$. For an incoming message, the first term would track the numerator in Equation (5) and the second term would track the denominator in Equation (5):

$$\mathbf{m}_{ji} = \left(\mathbf{v}_j, \exp(\mathbf{q}_i^T \mathbf{k}_j)\right) \tag{6}$$

Consequently, the commutative monoid that would aggregate messages is defined as:

$$(\mathbf{y}, \sigma) \oplus (\mathbf{y}', \sigma') = \left(\frac{\mathbf{y}\sigma + \mathbf{y}'\sigma'}{\sigma + \sigma'}, \sigma + \sigma'\right) \tag{7}$$

This equation ensures that message aggregation maintains commutativity and associativity with an identity element where there are no observed values and no terms to be normalised. Thus, the Transformer equation has been defined as a commutative monoid and can now be integrated into the standard MPNN equation as defined in Equation (1). However, MPNNs equipped with a fixed set of aggregation functions might struggle to emulate this complex commutative monoid when faced with OOD data. To fix this, Veličković (2023) suggests incorporating an attention mechanism into the MPNN framework, i.e. an attentional weighting function a is introduced to the update rule for node i as seen in Equation (1):

$$\mathbf{n}_i^{(l+1)} = \phi\left(\mathbf{n}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{n}_i, \mathbf{n}_j)\psi(\mathbf{n}_i, \mathbf{n}_j)\right) \tag{8}$$

[3] A commutative monoid is a set that is closed under a commutative associative binary operation and has an identity element.

where $a : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ is any attentional mechanism, such as Transformer or GATv2 (Brody et al., 2022) attention. Additionally, when this attention framework is coupled with a virtual node, such a model tends towards architectures such as Big Bird (Zaheer et al., 2020), which incorporates sparse attention mechanisms to reduce the quadratic complexity to linear.

2.3.3. GAT-MPNN

To investigate the effect of attention on the alignment between an MPNN and RT, we choose a from Equation (8) to be the GAT (Veličković et al., 2018) attention mechanism. This leads to a normalised attention coefficient for two adjoining nodes. Furthermore, we extend Equation (8) to include edge and graph features, resulting in Equation (9).

The un-normalised attention logit for adjoining nodes $\mathbf{n}_i$ and $\mathbf{n}_j$, a_{ij}, is defined in Equation (10). $\mathbf{W}_n$ and $\mathbf{W}_e$ are learnable weight matrices for the set of node and edge vectors, respectively. Similar to GAT (Veličković et al., 2018), we perform masked attention such that we only compute a_{ij} if and only if $j \in \mathcal{N}_i$. Equation (11) shows the normalisation of the attention logit, a_{ij}, by applying Softmax and LeakyReLU non-linearity (with negative slope $\alpha = 0.2$).

$$\mathbf{n}_i^{(l+1)} = \phi\left(\mathbf{n}_i, \bigoplus_{j \in \mathcal{N}_i} a(\mathbf{n}_i, \mathbf{n}_j, \mathbf{e}_{ij}, \mathbf{e}_{ji}, \mathbf{g})\psi(\mathbf{n}_i, \mathbf{n}_j, \mathbf{e}_{ij}, \mathbf{e}_{ji}, \mathbf{g})\right) \tag{9}$$

$$a_{ij} = \text{concat}(\mathbf{W}_n\mathbf{n}_i, \mathbf{W}_n\mathbf{n}_j, \mathbf{W}_e\mathbf{e}_{ij}, \mathbf{W}_e\mathbf{e}_{ji}, \mathbf{W}_g\mathbf{g}) \tag{10}$$

$$a(\mathbf{n}_i, \mathbf{n}_j, \mathbf{e}_{ij}, \mathbf{e}_{ji}, \mathbf{g}) = \frac{\exp(\text{LeakyReLU}(a_{ij}))}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(a_{ik}))} \tag{11}$$

3. Dataset Overview

Our experiments made use of the *CLRS Algorithmic Reasoning Benchmark* (Veličković et al., 2022). The CLRS Algorithmic Reasoning Benchmark is designed to assess machine learning models' ability to understand and execute fundamental algorithms. It includes 30 algorithms grouped into eight categories: sorting, searching, dynamic programming, greedy, strings, divide and conquer, graphs and geometry.

The specific algorithm we selected for this task was Jarvis' March. Jarvis' March forms part of the geometry category. The details of the algorithm can be found in Appendix B.

The CLRS benchmark is designed to test the OOD generalisation of neural networks (NNs). In the case of Jarvis' March, the benchmark creates test, validation and train sets with the configurations seen in Table 1. As shown in the table, test set is made up of notably larger graphs, allowing for the model of interest to be tested OOD.

Table 1: **Jarvis' March Dataset Description (Veličković et al., 2022)**

Dataset	Trajectories	Number of Nodes
Train	1000	16
Validation	32	16
Test	32	64

4. Methodology

Our methodology for aligning an MPNN with RT is as follows:

1. Train RT to perform algorithmic reasoning on Jarvis' March.
2. Freeze RT and curate a dataset of embeddings consisting of the inputs to RT and the corresponding generated output embeddings from each layer.
3. Train the MPNN and its variants to minimise the Mean Squared Error (MSE) between its and RT's embeddings.
4. Determine the degree of alignment between RT and the MPNN variants in-distribution and OOD by (1) measuring the MSE between the embeddings of the trained MPNN variants and the embeddings of RT and (2) taking a model distillation perspective, measuring the MPNN variant's performance on Jarvis' March.

4.1. Training the Relational Transformer

We train RT, as the best-performing Transformer model evaluated on *The CLRS Algorithmic Reasoning Benchmark* (Veličković et al., 2022) at the time, to execute the *Jarvis' March* (Jarvis, 1973) algorithm. The implementation from Diao and Loynd (2023) [4] was used to instantiate and train RT. Additionally, we enable the attention mechanism for the graph features. Graph feature attention is described in Equation (12). Subsequently, we incorporate the graph feature attention in Equation (3), resulting in Equation (13).

$$\mathbf{q}_g = \mathbf{W}_g^Q\mathbf{g}, \quad \mathbf{k}_g = \mathbf{W}_g^K\mathbf{g}, \quad \mathbf{v}_g = \mathbf{W}_g^V\mathbf{g} \tag{12}$$

$$\begin{aligned} \mathbf{q}_{ij} &= (\mathbf{W}_n^Q\mathbf{n}_i + \mathbf{W}_e^Q\mathbf{e}_{ji} + \mathbf{W}_g^Q\mathbf{g}), \\ \mathbf{k}_{ij} &= (\mathbf{W}_n^K\mathbf{n}_j + \mathbf{W}_e^K\mathbf{e}_{ji} + \mathbf{W}_g^K\mathbf{g}), \\ \mathbf{v}_{ij} &= (\mathbf{W}_n^V\mathbf{n}_j + \mathbf{W}_e^V\mathbf{e}_{ji} + \mathbf{W}_g^V\mathbf{g}) \end{aligned} \tag{13}$$

[4] https://github.com/CameronDiao/relational-transformer

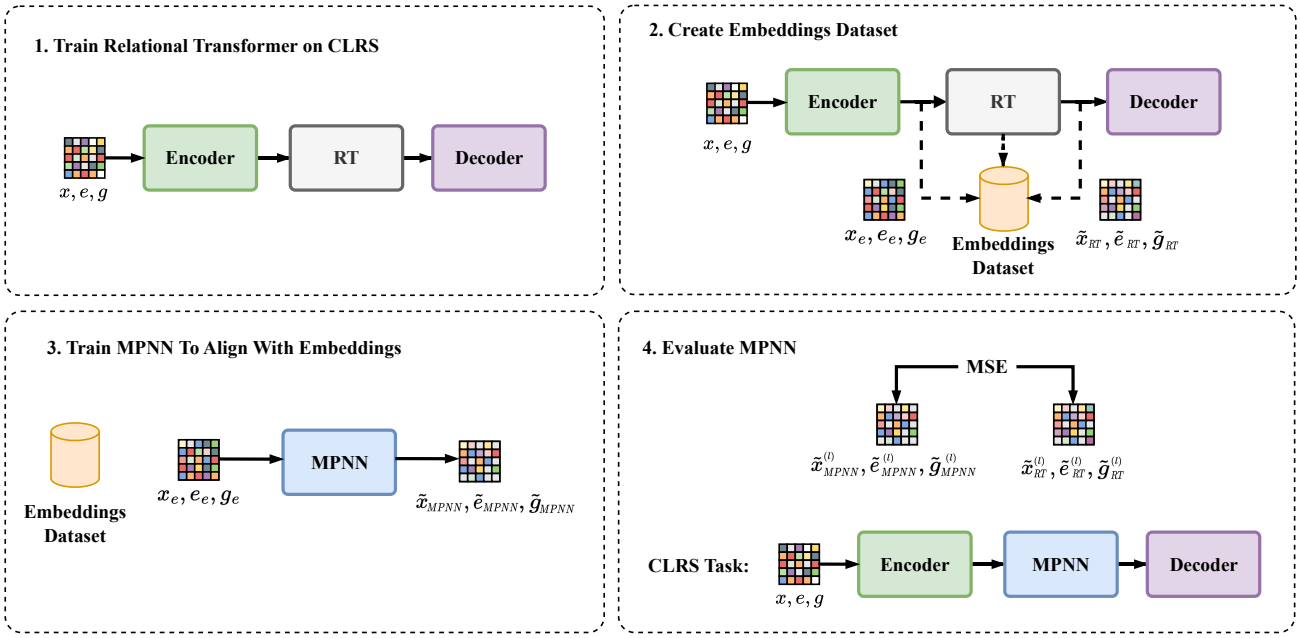

Figure 2: Alignment Pipeline. Each MPNN variant is trained to align with the intermediate embeddings of RT. Evaluation is then performed on an embeddings test set and the CLRS Task. x, e and g are the node, edge and graph features respectively.

We train RT for $5k$ optimisation steps and save the best model based on the validation loss. We use the default hyperparameters provided by Diao and Loynd (2023). This results in RT with three hidden layers, where each layer incorporates relational attention with 12 attention heads, with an attention head dimension size of 16 and a hidden dimension size of 192 between layers. We optimise RT with Adam (Kingma and Ba, 2017) with a learning rate of $2.5e^{-4}$. We ran all experiment configurations three times using three different seeds.

4.2. Curating the Embeddings Dataset

Keeping the same settings provided by Diao and Loynd (2023) and using the same CLRS train, validation and test splits in Section 4.1, we curate an embeddings dataset. The dataset consists of $10k$ train, 32 validation and 32 test samples. The code [4] from Diao and Loynd (2023) was modified to save the input and intermediate embeddings of RT. The inputs to RT consist of the adjacency matrix and the node, edge and graph features. The outputs from RT consist of the node and edge embeddings outputted from every RT layer.

4.3. Training MPNN Variants

We train the MPNN variants to minimise the MSE between their embeddings and the trained RT embeddings for each layer. The MSE loss for the node and edge embeddings can be seen in Equations (14) and (15) respectively. The combined loss can be seen in Equation (16). The settings of the MPNN can be seen in Table 2.

$$\mathcal{L}_{nodes} = \frac{1}{N}\sum_{l=1}^{L} ||\mathbf{X}^{l}_{\text{MPNN}} - \mathbf{X}^{l}_{\text{RT}}||_2^2 \tag{14}$$

$$\mathcal{L}_{edges} = \frac{1}{N^2}\sum_{l=1}^{L} ||\mathbf{E}^{l}_{\text{MPNN}} - \mathbf{E}^{l}_{\text{RT}}||_2^2 \tag{15}$$

$$\mathcal{L} = \mathcal{L}_{nodes} + \mathcal{L}_{edges} \tag{16}$$

Where $\mathbf{X} \in \mathbb{R}^{N \times f}$ denotes the matrix of node features, and N and f are the number of nodes and features respectively. $\mathbf{E} \in \mathbb{R}^{N^2 \times f}$ is the matrix of the edge features. L refers to the number of layers.

We perform experiments using a combination of all settings shown in Table 2. Using the best settings, which included layer normalisation [5], we train with 3 GAT-like attention heads. Additionally, we run experiments with a mid-dimension of 256 on the MPNN to determine whether over-parametrisation would improve alignment to RT. The mid-dimension of 256 is downsampled to match RTs hidden dimension of 192 using an MLP before being returned from a layer. We ran all experiment configurations three times with three different seeds and presented their results as a mean and standard deviation. We optimise each MPNN with Adam (Kingma and Ba, 2017) and use a learning rate of 0.001.

[5] As discussed in Section 5.5, we found that layer normalisation played an important role alignment.

Table 2: **Variations of MPNN**

Feature	Possible values
Virtual Node (VN)	True, False
Layer Normalisation (LN)	True, False
Mid Dimension	192, 256
Aggregation Function	sum, max
Localised Edge Updates (LEU)	True, False

4.4. Testing and Validation

We measure the MSE between the trained MPNN and RT embeddings on the validation and test sets. Additionally, we take a model distillation perspective and evaluate the MPNN variants, trained to minimise the MSE between embeddings, on Jarvis' March. However, one should note that the structure of the models evaluated on the CLRS benchmark consists of an *encoder*, *processor* and *decoder*. RT and the MPNN variants are instances of the *processor*. This paper aims to align the MPNN and RT, as the encoder and decoder were not part of the alignment process. Subsequently, the aligned MPNN used the trained RT model's encoder and decoder for this part of the evaluation.

5. Results & Discussion

In the following sections, we present the MSE losses for the MPNN trained on the embeddings dataset and the results of our distilled MPNN on Jarvis' March.

5.1. MPNN Alignment Based on MSE

Our initial approach was to train all MPNN variants on datasets created by trained RT with and without edge updates. However, the MPNN variant's losses did not differ significantly between these two conditions. Consequently, we only evaluated MPNNs on Jarvis' March, where RT performed edge updates. However, one can find the losses for the best-performing MPNN variants trained on RT without edge updates in Table 6.

For each MPNN variant, the aggregation function and the mid-dimension that yielded the best results based on validation loss are presented in Table 3. Table 7 shows the losses for all MPNN variants trained on RT with edge updates.

The results in Table 7 show that the test loss is consistently lower than both the training and validation losses in all cases. This occurs because the test embeddings are not OOD, unlike in the original CLRS task. This observation is confirmed by the poorer test set performance compared to the train and validation set performance on Jarvis' March, as shown in Table 4. These results are discussed in Section 5.9.

To visualise the alignment, Figure 3 illustrates the distribution of embeddings of the best performing MPNN variant (boldfaced in Table 3) compared to RT's for a randomly sampled batch. This visually shows the high degree of alignment between the MPNN's and RT's embedding distributions.

Furthermore, Table 3 shows the best-performing MPNN variants with GAT-like attention added. The results show that incorporating GAT-like attention does not yield significant alignment improvements in regard to the MSE loss.

5.2. MPNN Alignment Based on Jarvis' March

Following the training of all MPNN variants on the embeddings dataset, we evaluate each model on Jarvis' March as described in Section 4.4. We note that each MPNN variant was not directly trained to solve Jarvis' March but rather to align to the intermediate embeddings of RT. Thus, this experiment provides a more nuanced meaning to the alignment of the embeddings of the MPNN variants and RT. MSE measures how well the intermediate embeddings align, whilst performance on Jarvis' March describes how well each variant indirectly learns to solve the task. In addition, the MPNN variants adopt the trained RT's encoder and decoder from the CLRS pipeline. This is a much stricter setting in that minimal misalignment could result in poorer performance on the task.

For each MPNN variant, the aggregation function and the mid-dimension that yielded the best results are presented in Table 4. The table shows that the performance on Jarvis' March does not reflect the performance in terms of MSE, i.e. the performance on the task is significantly reduced compared to the trained RT performance. This suggests that embedding alignment does not necessarily translate to comparable performance on Jarvis' March.

In contrast to the test performance in terms of MSE shown in Table 3, where the test loss is lower than both the training and validation losses, the OOD test set performance on Jarvis' March follows a more typical pattern. Table 4 shows the test CLRS score is lower than the training and validation scores. This indicates that the MPNN struggles to perform effectively on the task itself when tested OOD.

5.3. Localised Edge Updates

As shown in Table 3, MPNN variants that perform localised edge updates (LEU) have a lower MSE loss than those that do not. This is expected as RT performs LEU. However, this performance gain does not hold true in our secondary experiment, where we evaluate the trained MPNNs on Jarvis' March. This again shows that a higher degree of alignment between the embeddings is not indicative of performance on the proxy task, as described in Section 5.2.

Table 3: **MSE loss of best performing MPNN variants trained on RT embeddings.**

Model Name	Agg Func	Mid Dim	Train Loss	Val Loss	Test Loss
MPNN	max	256	0.452 ± 0.000	0.489 ± 0.001	0.349 ± 0.000
MPNN + LN	max	256	0.405 ± 0.000	0.452 ± 0.000	0.301 ± 0.001
MPNN + VN	max	256	0.379 ± 0.003	0.402 ± 0.008	0.296 ± 0.001
MPNN + LEU	max	256	0.183 ± 0.000	0.261 ± 0.001	0.125 ± 0.004
MPNN + LN + VN	max	256	0.347 ± 0.000	0.362 ± 0.001	0.275 ± 0.001
MPNN + LN + LEU	max	192	0.151 ± 0.000	0.230 ± 0.000	0.118 ± 0.001
MPNN + VN + LEU	max	192	0.195 ± 0.147	0.172 ± 0.064	0.078 ± 0.000
MPNN + LN + VN + LEU	**max**	**256**	$\mathbf{0.070 \pm 0.001}$	$\mathbf{0.100 \pm 0.003}$	$\mathbf{0.057 \pm 0.000}$
MPNN + LN + ATT	max	256	0.405 ± 0.000	0.452 ± 0.000	0.302 ± 0.000
MPNN + LN + VN + ATT	max	256	0.347 ± 0.000	0.361 ± 0.001	0.273 ± 0.001
MPNN + LN + LEU + ATT	max	192	0.151 ± 0.000	0.231 ± 0.000	0.118 ± 0.001
MPNN + LN + VN + LEU + ATT	**max**	**256**	$\mathbf{0.073 \pm 0.006}$	$\mathbf{0.106 \pm 0.002}$	$\mathbf{0.057 \pm 0.004}$

Table 4: **CLRS Score of best performing MPNN variants trained on RT embeddings performance on Jarvis' March.** MPNNs were not trained on Jarvis' March; instead, they were trained to align to the intermediate embeddings of RT. Scores are from inference performed on each dataset.

Model Name	Agg Function	Mid Dim	Train Score (%)	Val Score (%)	Test Score (%)
RT	-	-	96.74 ± 1.50	98.21 ± 0.11	84.43 ± 3.42
MPNN	max	192	58.39 ± 0.35	57.97 ± 0.77	32.82 ± 0.09
MPNN + LN	sum	256	49.43 ± 0.01	51.37 ± 0.82	30.56 ± 0.24
MPNN + VN	max	192	59.24 ± 4.05	58.59 ± 4.81	33.05 ± 0.22
MPNN + LEU	max	256	46.62 ± 1.84	46.78 ± 2.57	33.73 ± 2.40
MPNN + LN + VN	sum	256	47.70 ± 0.48	47.47 ± 1.45	29.49 ± 0.62
MPNN + LN + LEU	sum	192	48.24 ± 0.35	46.84 ± 0.83	30.47 ± 0.34
MPNN + VN + LEU	**max**	**192**	$\mathbf{62.66 \pm 1.37}$	$\mathbf{62.07 \pm 1.44}$	$\mathbf{34.21 \pm 0.91}$
MPNN + LN + VN + LEU	sum	256	49.34 ± 1.98	50.61 ± 2.19	31.66 ± 1.62
MPNN + LN + ATT	max	256	40.53 ± 4.19	42.81 ± 3.20	28.87 ± 0.64
MPNN + LN + VN + ATT	max	256	42.59 ± 2.11	41.90 ± 3.24	29.28 ± 1.65
MPNN + LN + LEU + ATT	**max**	**192**	$\mathbf{45.47 \pm 1.09}$	$\mathbf{46.17 \pm 0.03}$	$\mathbf{29.35 \pm 0.20}$
MPNN + LN + VN + LEU + ATT	max	256	45.31 ± 1.46	44.88 ± 2.04	30.73 ± 0.04

To further illustrate the effect of LEUs, Figure 4 shows the heatmaps of the input node and edge features as well as the output edge features processed with and without LEU. The plot shows that with LEUs the processed edge embeddings are more expressive. Whereas, without edge updates the embeddings retain their original value.

5.4. Virtual Node

Table 7 shows that an MPNN with VN results in better MSE performance. This holds true for most experiments, reflecting the findings of Cai et al. (2023).

5.5. Layer Normalisation

All MPNN variants with layer normalisation show an improved validation and test loss when compared to their equivalent without layer normalisation. This result is expected as the RT model layer normalises the updated node features as the final operation in each layer. Thus, performing normalisation in the same way results in a better alignment between the MPNN and RT.

5.6. Attention

Adding GAT-like attention to our best-performing MPNN variants, in most cases, does not result in better alignment in terms of MSE. This result is surprising and suggests that equipping an MPNN with an attention mechanism on the

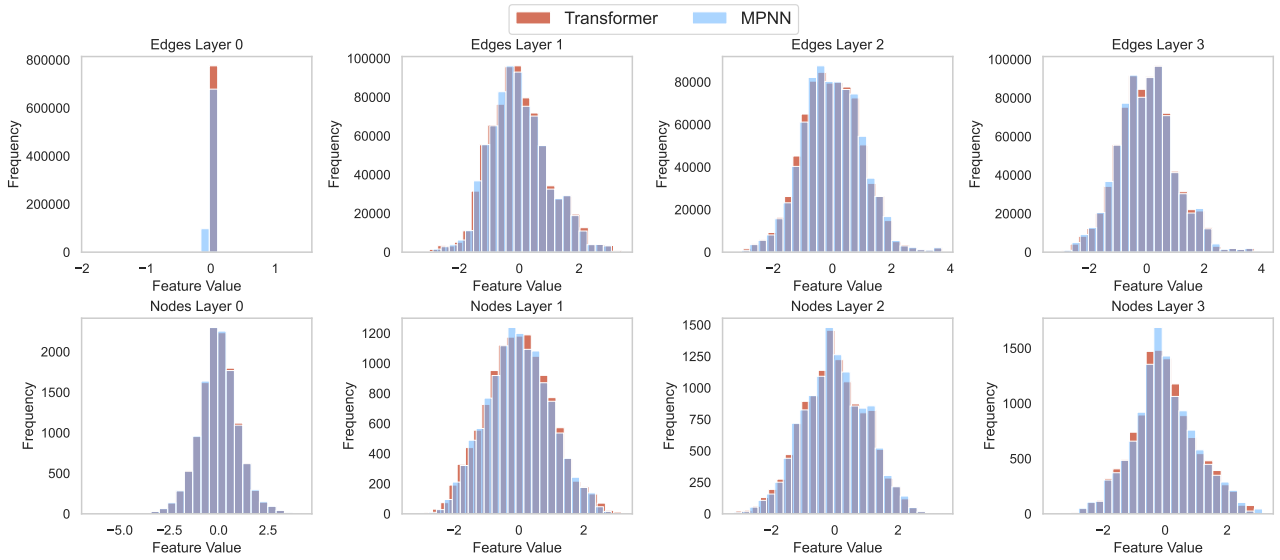

Figure 3: **MPNN + LN + VN + LEU.** Randomly sampled MPNN test set embeddings trained to align with RT embeddings.

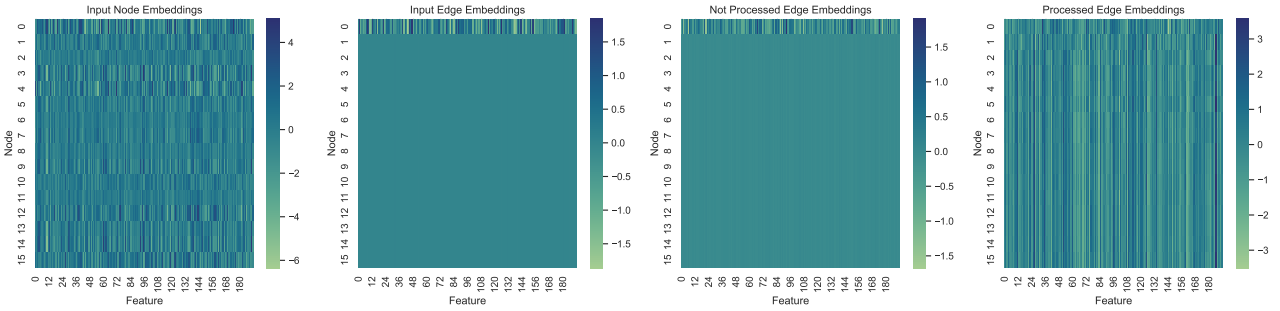

Figure 4: **Embedding heatmaps.** Left to right: 1) Input node features, 2) Input edge features, 3) Not processed edge embeddings 4) Processed edge embeddings.

task of aligning embeddings does not significantly improve its performance. However, a more thorough investigation, including the use of multiple heads and different attention mechanisms, would be needed to draw a meaningful conclusion on this result.

5.7. Mid Dimension & Over-Parametarisation

In order to investigate the effect of over-parametrisation on the task, the size of the mid-dimension was increased from 192 to 256. The majority of the experiments showed that there was no significant difference in performance between a specific MPNN variant that had a mid-dimension of 192 compared to 256. However, the MPNN variants with a mid-dimension of 256 often outperformed the 192 mid-dimension version of this model by ≈ 0.001. For this reason, they appear more frequently in Table 3.

5.8. Aggregation Function

Notably, MPNNs using *max* outperform those with *sum* in terms of MSE. This suggests that the *max* function may better capture relational features observed by RT for algorithmic reasoning tasks. This observation is consistent with prior research by Veličković et al. (2020) and Xu et al. (2020), indicating the improved performance of max aggregation in algorithmic reasoning tasks.

5.9. Out-of-Distribution Performance

Our results indicate that the OOD performance of the MPNN, i.e. the MSE performance on the test set, is better than in-distribution (train and validation). This is both unintuitive and contrary to the findings of Veličković et al. (2022), where they show MPNNs cannot achieve performance comparable with the training dataset on four times

larger graphs. However, in this case the MPNN is not directly trained on Jarvis' March. Instead, it uses the inputs to the processor (RT) of the CLRS benchmark and is trained to produce the same embeddings as RT. As a result of the transformation of the problem, the test set lies in the same distribution as the training and validation sets. Figures 3, 5 and 6 shows visual evidence of this. The plots show that for a given random sample, the distributions of the embeddings in the dataset between all splits (train, validation and test) are all within the range of $x \approx -5$ to 5 and follow a normal distribution.

6. Future Works

In the future, we would like to perform more exhaustive experiments to determine how much further we can align MPNNs and RT. This would include investigating more permutation invariant aggregation functions, such as *mean* and more settings of GAT-like attention, e.g. increasing the number of attention heads. Furthermore, we would investigate different forms of attention, such as the traditional transformer QKV method. We could extend our evaluations on Jarvis' March, and measure the number of fine-tuning optimisation steps needed for an MPNN to better align with RT. Additionally, the experiments could be performed on other tasks such as node or graph level tasks to ensure the reproduction of findings in these settings.

7. Conclusion

We have shown that we can align an MPNN with RT by training MPNN variants to produce embeddings that minimise the MSE between its embeddings and RT's. Subsequently, we illustrated the effect of various components, such as the virtual node, layer normalisation and attention on alignment, by exhaustively training MPNNs with all permutations of these components.

We indicate the three most important components are the aggregation function, the virtual node and layer normalisation. Our results show that MPNN variants equipped with a virtual node, the *max* aggregation function and layer normalisation aligned best with RT.

Based on our results, the MPNN was able to perform better on the "OOD" test set than the "in-distribution" validation and train sets. We determined through visualisation that the test set was in-distribution in the embeddings dataset and this resulted in the improved performance.

Evaluating the aligned MPNNs on Jarvis' March allowed us to interpret the alignment of the MPNN and RT from a model distillation perspective. However, this is a much stricter setting, where minimal misalignment would result in poor performance due to adopting RT's trained encoder and decoder. This showed that although the MPNNs converged to a low MSE, indicating strong alignment, this did not necessarily translate to comparable performance on the proxy task.

8. Acknowledgements

We would like to thank Simon Osindero (DeepMind) and Andrea Banino (DeepMind) for their insightful review and constructive suggestions.

References

Xiao Li, Li Sun, Mengjie Ling, and Yan Peng. A survey of graph neural network based recommendation in social networks. *Neurocomputing*, 549:126441, 2023. doi: 10.1016/J.NEUCOM.2023.126441. URL https://doi.org/10.1016/j.neucom.2023.126441.

Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Comput. Surv.*, 55(5), dec 2022. ISSN 0360-0300. doi: 10.1145/3535101.

Kehang Han, Balaji Lakshminarayanan, and Jeremiah Zhe Liu. Reliable graph neural networks for drug discovery under distributional shift. In *NeurIPS 2021 Workshop on Distribution Shifts: Connecting Methods and Applications*, 2021.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 06–11 Aug 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs, 2021.

Cameron Diao and Ricky Loynd. Relational attention: Generalizing transformers for graph-structured tasks. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Andrew J Dudzik and Petar Veličković. Graph neural networks are dynamic programmers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 20635–20647. Curran Associates, Inc., 2022.

Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Veličković. A generalist neural algorithmic learner. In Bastian Rieck and Razvan Pascanu, editors, *Proceedings of the First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, pages 2:1–2:23. PMLR, 09–12 Dec 2022.

Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between MPNN and graph transformer. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 3408–3430. PMLR, 23–29 Jul 2023.

Eran Rosenbluth, Jan Tönshoff, Martin Ritzert, Berke Kisin, and Martin Grohe. Distinguished in uniform: Self-attention vs. virtual nodes. In *The Twelfth International Conference on Learning Representations*, 2024.

Petar Veličković. Mpnn alignment monoids. Tweet, 2023. Twitter post.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17283–17297. Curran Associates, Inc., 2020.

Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The CLRS algorithmic reasoning benchmark. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22084–22102. PMLR, 17–23 Jul 2022.

R.A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973. ISSN 0020-0190.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Petar Veličković, Lars Buesing, Matthew Overlan, Razvan Pascanu, Oriol Vinyals, and Charles Blundell. Pointer graph networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2232–2244. Curran Associates, Inc., 2020.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Du, Ken ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations*, 2020.

A. Localised edge updates in RT

The process for calculating the edge updates described by the edge update function, ϕ_e, can be seen in Equations (17) to (19).

$$\mathbf{m}_{lij} = \text{ReLU}\left(\mathbf{W}^4\left(\text{concat}\left(\mathbf{e}_{ij}^{(l)}, \mathbf{e}_{ji}^{(l)}, \mathbf{n}_i^{(l+1)}, \mathbf{n}_j^{(l+1)}\right)\right)\right) \quad (17)$$

where $\mathbf{e}_{ji}^{(l)}$ is the edge vector from node j to node i at layer l, $\mathbf{n}_i^{(l+1)}$ and $\mathbf{n}_j^{(l+1)}$ are the updated node vectors after self-attention at layer $l+1$, $\mathbf{W}^4 \in \mathbb{R}^{(2d_e+2d_n)\times d_{eh1}}$ is a weight matrix that transforms the concatenated vector into a hidden representation, where d_e is the dimension of the edge vector, d_n is the dimension of the node vector, and d_{eh1} is the size of the hidden layer and $\mathbf{m}_{lij}$ is the aggregated message for the edge $\mathbf{e}_{ji}$ at layer l.

$$\mathbf{u}_{lij} = \text{LayerNorm}\left(\mathbf{W}^5 \mathbf{m}_{lij} + \mathbf{e}_{ji}^{(l)}\right) \tag{18}$$

$$\mathbf{e}_{ji}^{(l+1)} = \text{LayerNorm}\left(\text{ReLU}\left(\mathbf{W}^7\left(\mathbf{W}^6 \mathbf{u}_{lij}\right)\right) + \mathbf{u}_{lij}\right) \tag{19}$$

where $\mathbf{W}^5 \in \mathbb{R}^{d_{eh1} \times d_e}$, $\mathbf{W}^6 \in \mathbb{R}^{d_e \times d_{eh2}}$, $\mathbf{W}^7 \in \mathbb{R}^{d_{eh2} \times d_e}$ are the weight matrices for the feed-forward network within the edge update mechanism, with d_{eh2} being the size of the second hidden layer, $\mathbf{u}_{lij}$ is the intermediate update vector for the edge $\mathbf{e}_{ji}$ at layer l and $\mathbf{e}_{ji}^{(l+1)}$ is the updated edge vector after applying the edge update function.

B. Jarvis' March

Jarvis' March is used to find the convex hull in a set of points. The algorithm starts at the leftmost point, which is part of the convex hull, and iteratively selects the point that makes the most counter-clockwise turn with respect to the previous point. This process continues until the algorithm returns to the starting point, which completes the convex hull. (Jarvis, 1973)

C. Extended results

Table 5: **Number of model parameters for MPNN variants**

Model	Mid Dim	Number of Parameters
RT	-	4,131,034
MPNN	192	1,037,184
MPNN	256	1,259,136
MPNN + LN	192	1,038,336
MPNN + LN	256	1,260,288
MPNN + VN	192	1,037,184
MPNN + VN	256	1,259,136
MPNN + LEU	192	1,105,224
MPNN + LEU	256	1,340,616
MPNN + LN + VN	192	1,038,336
MPNN + LN + VN	256	1,260,288
MPNN + LN + LEU	192	1,106,376
MPNN + LN + LEU	256	1,341,768
MPNN + VN + LEU	192	1,105,224
MPNN + VN + LEU	256	1,340,616
MPNN + LN + VN + LEU	192	1,106,376
MPNN + LN + VN + LEU	256	1,341,768
MPNN + ATT	192	1,044,132
MPNN + ATT	256	1,266,084
MPNN + LN + ATT	192	1,045,284
MPNN + LN + ATT	256	1,267,236
MPNN + VN + ATT	192	1,044,132
MPNN + VN + ATT	256	1,266,084
MPNN + LEU + ATT	192	1,112,172
MPNN + LEU + ATT	256	1,347,564
MPNN + LN + VN + ATT	192	1,045,284
MPNN + LN + VN + ATT	256	1,267,236
MPNN + LN + LEU + ATT	192	1,113,324
MPNN + LN + LEU + ATT	256	1,348,716
MPNN + VN + LEU + ATT	192	1,112,172
MPNN + VN + LEU + ATT	256	1,347,564
MPNN + LN + VN + LEU + ATT	192	1,113,324
MPNN + LN + VN + LEU + ATT	256	1,348,716

Table 6: **Best-performing MPNN variants trained on RT embeddings without LEU**

Model Name	Agg Func	Mid Dim	Train Loss	Val Loss	Test Loss
MPNN	max	192	0.134 ± 0.000	0.147 ± 0.000	0.095 ± 0.001
MPNN + LN	max	256	0.091 ± 0.000	0.110 ± 0.000	0.055 ± 0.001
MPNN + VN	max	192	0.091 ± 0.031	0.086 ± 0.018	0.055 ± 0.000
MPNN + LEU	max	192	0.112 ± 0.000	0.126 ± 0.000	0.162 ± 0.011
MPNN + LN + VN	max	256	0.044 ± 0.000	0.052 ± 0.001	0.032 ± 0.001
MPNN + LN + LEU	max	256	0.088 ± 0.000	0.106 ± 0.000	0.141 ± 0.001
MPNN + VN + LEU	max	192	0.057 ± 0.000	0.060 ± 0.001	0.051 ± 0.003
MPNN + LN + VN + LEU	**max**	**256**	$\mathbf{0.038 \pm 0.000}$	$\mathbf{0.046 \pm 0.001}$	$\mathbf{0.028 \pm 0.000}$

Table 7: **Full experiments table MPNN variants trained on RT embeddings with LEU - MSE Loss.**

Model Name	Agg Func	Mid Dim	Train Loss	Val Loss	Test Loss
MPNN	sum	192	0.477 ± 0.002	0.508 ± 0.000	0.368 ± 0.001
MPNN	max	192	0.451 ± 0.001	0.490 ± 0.003	0.350 ± 0.002
MPNN	sum	256	0.479 ± 0.000	0.509 ± 0.001	0.368 ± 0.002
MPNN	**max**	**256**	$\mathbf{0.452 \pm 0.000}$	$\mathbf{0.489 \pm 0.001}$	$\mathbf{0.349 \pm 0.000}$
MPNN + LN	sum	192	0.412 ± 0.000	0.464 ± 0.001	0.298 ± 0.001
MPNN + LN	max	192	0.406 ± 0.002	0.453 ± 0.002	0.302 ± 0.003
MPNN + LN	sum	256	0.412 ± 0.000	0.464 ± 0.000	0.299 ± 0.000
MPNN + LN	**max**	**256**	$\mathbf{0.405 \pm 0.000}$	$\mathbf{0.452 \pm 0.000}$	$\mathbf{0.301 \pm 0.001}$
MPNN + VN	sum	192	0.487 ± 0.001	0.488 ± 0.000	0.317 ± 0.001
MPNN + VN	max	192	0.422 ± 0.001	0.415 ± 0.002	0.297 ± 0.001
MPNN + VN	sum	256	0.488 ± 0.002	0.488 ± 0.003	0.318 ± 0.001
MPNN + VN	**max**	**256**	$\mathbf{0.379 \pm 0.003}$	$\mathbf{0.402 \pm 0.008}$	$\mathbf{0.296 \pm 0.001}$
MPNN + LEU	sum	192	0.195 ± 0.001	0.273 ± 0.001	0.130 ± 0.003
MPNN + LEU	max	192	0.183 ± 0.000	0.263 ± 0.003	0.126 ± 0.003
MPNN + LEU	sum	256	0.199 ± 0.003	0.279 ± 0.001	0.133 ± 0.002
MPNN + LEU	**max**	**256**	$\mathbf{0.183 \pm 0.000}$	$\mathbf{0.261 \pm 0.001}$	$\mathbf{0.125 \pm 0.004}$
MPNN + LN + VN	sum	192	0.400 ± 0.002	0.443 ± 0.002	0.288 ± 0.002
MPNN + LN + VN	max	192	0.347 ± 0.000	0.362 ± 0.001	0.272 ± 0.001
MPNN + LN + VN	sum	256	0.400 ± 0.000	0.442 ± 0.000	0.288 ± 0.000
MPNN + LN + VN	**max**	**256**	$\mathbf{0.347 \pm 0.000}$	$\mathbf{0.362 \pm 0.001}$	$\mathbf{0.275 \pm 0.001}$
MPNN + LN + LEU	sum	192	0.161 ± 0.001	0.246 ± 0.001	0.114 ± 0.001
MPNN + LN + LEU	**max**	**192**	$\mathbf{0.151 \pm 0.000}$	$\mathbf{0.230 \pm 0.000}$	$\mathbf{0.118 \pm 0.001}$
MPNN + LN + LEU	sum	256	0.161 ± 0.001	0.247 ± 0.001	0.114 ± 0.002
MPNN + LN + LEU	max	256	0.152 ± 0.001	0.231 ± 0.002	0.120 ± 0.001
MPNN + VN + LEU	sum	192	0.215 ± 0.001	0.304 ± 0.002	0.137 ± 0.004
MPNN + VN + LEU	**max**	**192**	$\mathbf{0.195 \pm 0.147}$	$\mathbf{0.172 \pm 0.064}$	$\mathbf{0.078 \pm 0.000}$
MPNN + VN + LEU	sum	256	0.216 ± 0.002	0.299 ± 0.003	0.125 ± 0.001
MPNN + VN + LEU	max	256	0.264 ± 0.003	0.197 ± 0.002	0.083 ± 0.002
MPNN + LN + VN + LEU	sum	192	0.114 ± 0.002	0.168 ± 0.001	0.078 ± 0.007
MPNN + LN + VN + LEU	max	192	0.070 ± 0.000	0.100 ± 0.001	0.054 ± 0.000
MPNN + LN + VN + LEU	sum	256	0.116 ± 0.002	0.171 ± 0.003	0.079 ± 0.000
MPNN + LN + VN + LEU	**max**	**256**	$\mathbf{0.070 \pm 0.001}$	$\mathbf{0.100 \pm 0.003}$	$\mathbf{0.057 \pm 0.000}$

Table 8: **Full experiments table MPNN variants trained on RT embeddings with LEU performance on CLRS Task**

Model Name	Mid Dim	Agg Function	Train Score	Val Score	Test Score
MPNN	sum	192	25.97 ± 2.92	23.30 ± 3.18	25.56 ± 4.26
MPNN	**max**	**192**	$\mathbf{58.39 \pm 0.35}$	$\mathbf{57.97 \pm 0.77}$	$\mathbf{32.82 \pm 0.09}$
MPNN	sum	256	38.21 ± 6.96	36.45 ± 8.18	34.36 ± 0.55
MPNN	max	256	57.74 ± 2.04	56.89 ± 0.29	33.99 ± 1.41
MPNN + LN	sum	192	49.41 ± 0.02	51.33 ± 0.34	30.45 ± 0.22
MPNN + LN	max	192	42.08 ± 0.65	44.36 ± 0.58	29.04 ± 0.73
MPNN + LN	**sum**	**256**	$\mathbf{49.43 \pm 0.01}$	$\mathbf{51.37 \pm 0.82}$	$\mathbf{30.56 \pm 0.24}$
MPNN + LN	max	256	41.22 ± 1.70	44.06 ± 0.32	28.86 ± 0.23
MPNN + VN	sum	192	46.02 ± 1.92	47.61 ± 0.85	31.32 ± 0.51
MPNN + VN	**max**	**192**	$\mathbf{59.24 \pm 4.05}$	$\mathbf{58.59 \pm 4.81}$	$\mathbf{33.05 \pm 0.22}$
MPNN + VN	sum	256	49.91 ± 1.36	48.84 ± 0.77	32.66 ± 0.78
MPNN + VN	max	256	49.07 ± 20.23	47.87 ± 21.34	29.55 ± 5.51
MPNN + LEU	sum	192	44.83 ± 1.26	45.74 ± 0.47	31.84 ± 0.06
MPNN + LEU	max	192	43.18 ± 1.93	42.30 ± 3.26	31.43 ± 2.33
MPNN + LEU	sum	256	41.64 ± 4.19	41.51 ± 4.38	28.10 ± 4.97
MPNN + LEU	**max**	**256**	$\mathbf{46.62 \pm 1.84}$	$\mathbf{46.78 \pm 2.57}$	$\mathbf{33.73 \pm 2.40}$
MPNN + LN + VN	sum	192	47.30 ± 0.11	46.89 ± 0.31	29.70 ± 0.44
MPNN + LN + VN	max	192	43.33 ± 0.46	42.74 ± 0.90	29.33 ± 0.98
MPNN + LN + VN	**sum**	**256**	$\mathbf{47.70 \pm 0.48}$	$\mathbf{47.47 \pm 1.45}$	$\mathbf{29.49 \pm 0.62}$
MPNN + LN + VN	max	256	41.50 ± 0.35	41.87 ± 0.10	28.75 ± 1.33
MPNN + LN + LEU	**sum**	**192**	$\mathbf{48.24 \pm 0.35}$	$\mathbf{46.84 \pm 0.83}$	$\mathbf{30.47 \pm 0.34}$
MPNN + LN + LEU	max	192	45.58 ± 1.79	46.21 ± 1.01	30.54 ± 1.04
MPNN + LN + LEU	sum	256	44.18 ± 2.38	43.95 ± 3.12	29.25 ± 0.64
MPNN + LN + LEU	max	256	45.82 ± 0.88	46.65 ± 0.18	29.57 ± 0.52
MPNN + VN + LEU	sum	192	40.48 ± 2.10	38.90 ± 2.11	30.74 ± 0.73
MPNN + VN + LEU	**max**	**192**	$\mathbf{62.66 \pm 1.37}$	$\mathbf{62.07 \pm 1.44}$	$\mathbf{34.21 \pm 0.91}$
MPNN + VN + LEU	sum	256	44.27 ± 0.51	43.43 ± 2.55	30.75 ± 0.81
MPNN + VN + LEU	max	256	45.99 ± 19.12	45.93 ± 18.46	29.27 ± 3.57
MPNN + LN + VN + LEU	sum	192	43.79 ± 0.25	44.34 ± 0.53	30.89 ± 0.11
MPNN + LN + VN + LEU	max	192	43.69 ± 1.70	43.47 ± 1.44	29.55 ± 1.17
MPNN + LN + VN + LEU	**sum**	**256**	$\mathbf{49.34 \pm 1.98}$	$\mathbf{50.61 \pm 2.19}$	$\mathbf{31.66 \pm 1.62}$
MPNN + LN + VN + LEU	max	256	43.53 ± 0.06	43.40 ± 0.39	30.09 ± 0.22

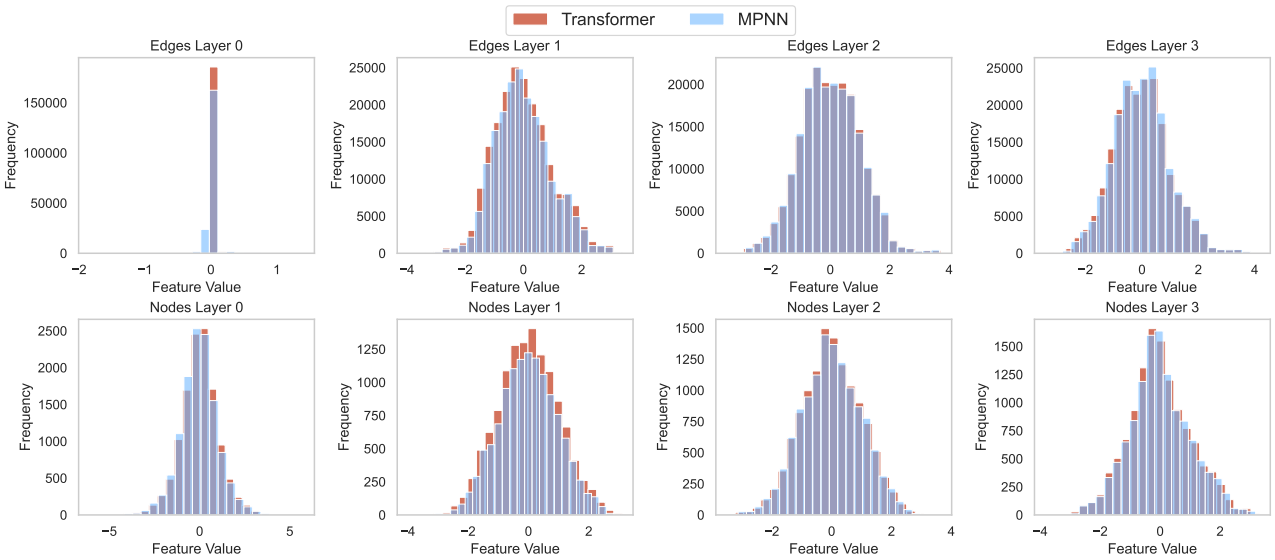

Figure 5: **MPNN + LN + VN + LEU.** Randomly sampled MPNN training set embeddings trained to align with RT embeddings.

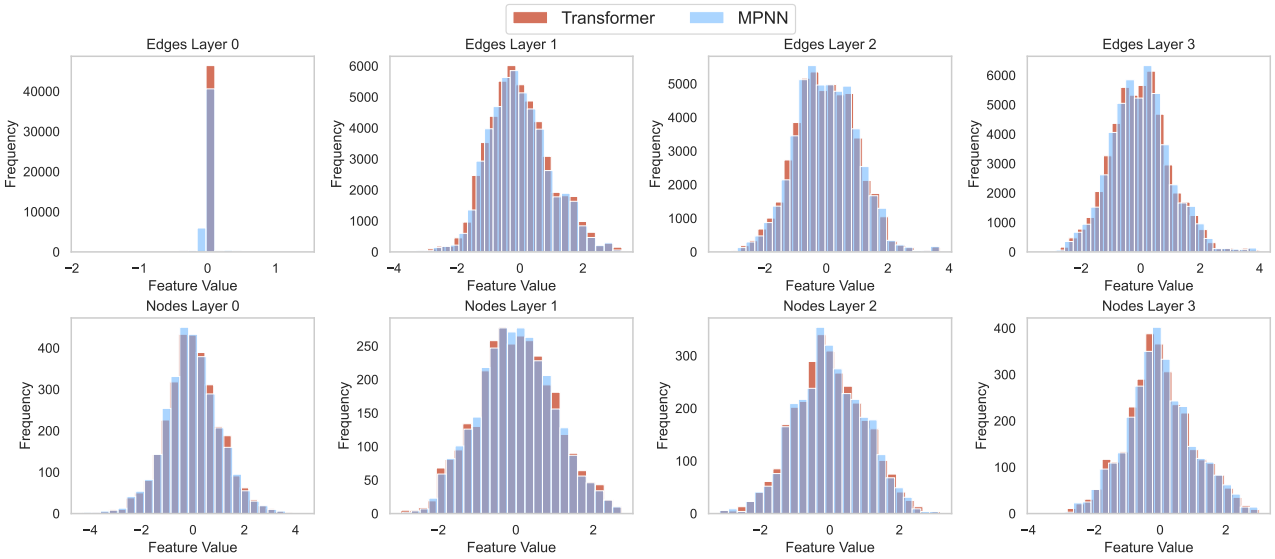

Figure 6: **MPNN + LN + VN + LEU.** Randomly sampled MPNN validation set embeddings trained to align with RT embeddings.