On the Connection Between MPNN and Graph Transformer

Chen Cai [1] **Truong Son Hy** [1] **Rose Yu** [1] **Yusu Wang** [1]

Abstract

Graph Transformer (GT) recently has emerged as a new paradigm of graph learning algorithms, outperforming the previously popular Message Passing Neural Network (MPNN) on multiple benchmarks. Previous work (Kim et al., 2022) shows that with proper position embedding, GT can approximate MPNN arbitrarily well, implying that GT is at least as powerful as MPNN. In this paper, we study the inverse connection and show that MPNN with virtual node (VN), a commonly used heuristic with little theoretical understanding, is powerful enough to arbitrarily approximate the self-attention layer of GT. In particular, we first show that if we consider one type of linear transformer, the so-called Performer/Linear Transformer (Choromanski et al., 2020; Katharopoulos et al., 2020b), then MPNN + VN with only $\mathcal{O}(1)$ depth and $\mathcal{O}(1)$ width can approximate a self-attention layer in Performer/Linear Transformer. Next, via a connection between MPNN + VN and DeepSets, we prove the MPNN + VN with $\mathcal{O}(n^d)$ width and $\mathcal{O}(1)$ depth can approximate the self-attention layer arbitrarily well, where d is the input feature dimension. Lastly, under some assumptions, we provide an explicit construction of MPNN + VN with $\mathcal{O}(1)$ width and $\mathcal{O}(n)$ depth approximating the self-attention layer in GT arbitrarily well. On the empirical side, we demonstrate that 1) MPNN + VN is a surprisingly strong baseline, outperforming GT on the recently proposed Long Range Graph Benchmark (LRGB) dataset, 2) our MPNN + VN improves over early implementation on a wide range of OGB datasets and 3) MPNN + VN outperforms Linear Transformer and MPNN on the climate modeling task.

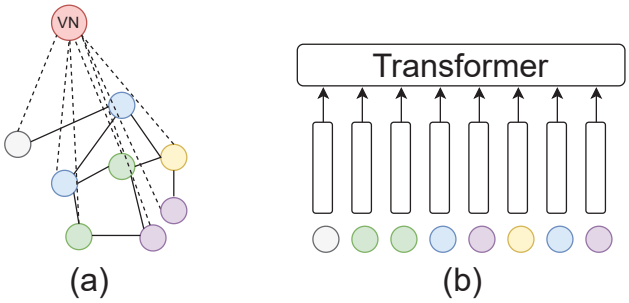

Figure 1: MPNN + VN and Graph Transformers.

1. Introduction

MPNN (Message Passing Neural Network) (Gilmer et al., 2017) has been the leading architecture for processing graph-structured data. Recently, transformers in natural language processing (Vaswani et al., 2017; Kalyan et al., 2021) and vision (d'Ascoli et al., 2021; Han et al., 2022) have extended their success to the domain of graphs. There have been several pieces of work (Ying et al., 2021; Wu et al., 2021; Kreuzer et al., 2021; Rampášek et al., 2022; Kim et al., 2022) showing that with careful position embedding (Lim et al., 2022), graph transformers (GT) can achieve compelling empirical performances on large-scale datasets and start to challenge the dominance of MPNN.

MPNN imposes a sparsity pattern on the computation graph and therefore enjoys linear complexity. It however suffers from well-known over-smoothing (Li et al., 2018; Oono & Suzuki, 2019; Cai & Wang, 2020) and over-squashing (Alon & Yahav, 2020; Topping et al., 2021) issues, limiting its usage on long-range modeling tasks where the label of one node depends on features of nodes far away. GT relies purely on position embedding to encode the graph structure and uses vanilla transformers on top. [1] It models all pairwise interactions directly in one layer, making it computationally more expensive. Compared to MPNN, GT shows promising results on tasks where modeling long-range interaction is the key, but the quadratic complexity of self-attention in GT

[1]University of California San Diego, San Diego, USA. Correspondence to: Chen Cai <c1cai@ucsd.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[1]GT in this paper refers to the practice of tokenizing graph nodes and applying standard transformers on top (Ying et al., 2021; Kim et al., 2022). There exists a more sophisticated GT (Kreuzer et al., 2021) that further conditions attention on edge types but it is not considered in this paper.

Table 1: Summary of approximation result of MPNN + VN on self-attention layer. n is the number of nodes and d is the feature dimension of node features. The dependency on d is hidden.

	Depth	Width	Self-Attention	Note
Theorem 4.1	$\mathcal{O}(1)$	$\mathcal{O}(1)$	Approximate	Approximate self attention in Performer (Choromanski et al., 2020)
Theorem 5.5	$\mathcal{O}(1)$	$\mathcal{O}(n^d)$	Full	Leverage the universality of equivariant DeepSets
Theorem 6.3	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Full	Explicit construction, strong assumption on $\mathcal{X}$
Proposition B.9	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Full	Explicit construction, more relaxed (but still strong) assumption on $\mathcal{X}$

limits its usage to graphs of medium size. Scaling up GT to large graphs remains an active research area (Wu et al., 2022).

Theoretically, it has been shown that graph transformers can be powerful graph learners (Kim et al., 2022), i.e., graph transformers with appropriate choice of token embeddings have the capacity of approximating linear permutation equivariant basis, and therefore can approximate 2-IGN (Invariant Graph Network), a powerful architecture that is at least as expressive as MPNN (Maron et al., 2018). This raises an important question that *whether GT is strictly more powerful than MPNN*. Can we approximate GT with MPNN?

One common intuition of the advantage of GT over MPNN is its ability to model long-range interaction more effectively. However, from the MPNN side, one can resort to a simple trick to escape locality constraints for effective long-range modeling: the use of an additional *virtual node (VN)* that connects to all input graph nodes. On a high level, MPNN + VN augments the existing graph with one virtual node, which acts like global memory for every node exchanging messages with other nodes. Empirically this simple trick has been observed to improve the MPNN and has been widely adopted (Gilmer et al., 2017; Hu et al., 2020; 2021) since the early beginning of MPNN (Gilmer et al., 2017; Battaglia et al., 2018). However, there is very little theoretical study of MPNN + VN (Hwang et al., 2022).

In this work, we study the theoretical property of MPNN + VN, and its connection to GT. We systematically study the representation power of MPNN + VN, both for certain approximate self-attention and for the full self-attention layer, and provide a depth-width trade-off, summarized in Table 1. In particular,

- With $\mathcal{O}(1)$ depth and $\mathcal{O}(1)$ width, MPNN + VN can approximate one self-attention layer of Performer (Choromanski et al., 2020) and Linear Transformer (Katharopoulos et al., 2020b), a type of linear transformers (Tay et al., 2020).
- Via a link between MPNN + VN with DeepSets (Zaheer et al., 2017), we prove MPNN + VN with $\mathcal{O}(1)$ depth and $\mathcal{O}(n^d)$ width (d is the input feature dimension) is permutation equivariant universal, implying it can approximate self-attention layer and even full-transformers.
- Under certain assumptions on node features, we prove an explicit construction of $\mathcal{O}(n)$ depth $\mathcal{O}(1)$ width MPNN + VN approximating 1 self-attention layer arbitrarily well on graphs of size n. Unfortunately, the assumptions on node features are rather strong, and whether we can alleviate them will be an interesting future direction to explore.
- Empirically, we show 1) that MPNN + VN works surprisingly well on the recently proposed LRGB (long-range graph benchmarks) datasets (Dwivedi et al., 2022), which arguably require long-range interaction reasoning to achieve strong performance 2) our implementation of MPNN + VN is able to further improve the early implementation of MPNN + VN on OGB datasets and 3) MPNN + VN outperforms Linear Transformer (Katharopoulos et al., 2020b) and MPNN on the climate modeling task.

2. Related Work

Virtual node in MPNN. The virtual node augments the graph with an additional node to facilitate the information exchange among all pairs of nodes. It is a heuristic proposed in (Gilmer et al., 2017) and has been observed to improve the performance in different tasks (Hu et al., 2021; 2020). Surprisingly, its theoretical properties have received little study. To the best of our knowledge, only a recent paper (Hwang et al., 2022) analyzed the role of the virtual node in the link prediction setting in terms of 1) expressiveness of the learned link representation and 2) the potential impact on under-reaching and over-smoothing.

Graph transformer. Because of the great successes of Transformers in natural language processing (NLP) (Vaswani et al., 2017; Wolf et al., 2020) and recently in computer vision (Dosovitskiy et al., 2020; d'Ascoli et al., 2021; Liu et al., 2021), there is great interest in extending transformers for graphs (Müller et al., 2023). One common belief of advantage of graph transformer over MPNN is its capacity in capturing long-range interactions while alleviating over-smoothing (Li et al., 2018; Oono & Suzuki, 2019;

Cai & Wang, 2020) and over-squashing in MPNN (Alon & Yahav, 2020; Topping et al., 2021).

Fully-connected Graph transformer (Dwivedi & Bresson, 2020) was introduced with eigenvectors of graph Laplacian as the node positional encoding (PE). Various follow-up works proposed different ways of PE to improve GT, ranging from an invariant aggregation of Laplacian's eigenvectors in SAN (Kreuzer et al., 2021), pair-wise graph distances in Graphormer (Ying et al., 2021), relative PE derived from diffusion kernels in GraphiT (Mialon et al., 2021), and recently Sign and Basis Net (Lim et al., 2022) with a principled way of handling sign and basis invariance. Other lines of research in GT include combining MPNN and GT (Wu et al., 2021; Rampášek et al., 2022), encoding the substructures (Chen et al., 2022), GT for directed graphs (Geisler et al., 2023), and efficient graph transformers for large graphs (Wu et al., 2022).

Deep Learning on Sets. Janossy pooling (Murphy et al., 2018) is a framework to build permutation invariant architecture for sets using permuting & averaging paradigm while limiting the number of elements in permutations to be $k < n$. Under this framework, DeepSets (Zaheer et al., 2017) and PointNet (Qi et al., 2017) are recovered as the case of $k = 1$. For case $k = 2$, self-attention and Relation Network (Santoro et al., 2017) are recovered (Wagstaff et al., 2022). Although DeepSets and Relation Network (Santoro et al., 2017) are both shown to be universal permutation invariant, recent work (Zweig & Bruna, 2022) provides a finer characterization on the representation gap between the two architectures.

3. Preliminaries

We denote $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ the concatenation of graph node features and positional encodings, where node i has feature $\boldsymbol{x}_i \in \mathbb{R}^d$. When necessary, we use $\boldsymbol{x}_j^{(l)}$ to denote the node j's feature at depth l. Let $\mathcal{M}$ be the space of multisets of vectors in $\mathbb{R}^d$. We use $\mathcal{X} \subseteq \mathbb{R}^{n\times d}$ to denote the space of node features and the $\mathcal{X}_i$ be the projection of $\mathcal{X}$ on i-th coordinate. $\|\cdot\|$ denotes the 2-norm. $[\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}]$ denotes the concatenation of $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$. $[n]$ stands for the set $\{1, 2, ..., n\}$.

Definition 3.1 (attention)**.** We denote key and query matrix as $\boldsymbol{W}_K, \boldsymbol{W}_Q \in \mathbb{R}^{d\times d'}$, and value matrix as $\boldsymbol{W}_V \in \mathbb{R}^{d\times d}$ [2]. Attention score between two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^{d\times 1}$ is defined as $\alpha(\boldsymbol{u}, \boldsymbol{v}) = \text{softmax}(\boldsymbol{u}^T\boldsymbol{W}_Q(\boldsymbol{W}_K)^T\boldsymbol{v})$. We denote $\mathcal{A}$ as the space of attention α for different $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$. We also define unnormalized attention score $\alpha'(\cdot,\cdot)$ to be $\alpha'(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^T\boldsymbol{W}_Q(\boldsymbol{W}_K)^T\boldsymbol{v}$. Self attention layer is a matrix function $\boldsymbol{L} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ of the following form: $\boldsymbol{L}(\boldsymbol{X}) = \text{softmax}(\boldsymbol{X}\boldsymbol{W}_Q(\boldsymbol{X}\boldsymbol{W}_K)^T)\boldsymbol{X}\boldsymbol{W}_V$.

[2]For simplicity, we assume the output dimension of self-attention is the same as the input dimension. All theoretical results can be extended to the case where the output dimension is different from d.

3.1. MPNN Layer

Definition 3.2 (MPNN layer (Gilmer et al., 2017))**.** An MPNN layer on a graph G with node features $\boldsymbol{x}^{(k)}$ at k-th layer and edge features $\boldsymbol{e}$ is of the following form

$$\boldsymbol{x}_i^{(k)} = \gamma^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in\mathcal{N}(i)}\phi^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right)$$

Here $\gamma : \mathbb{R}^d \times \mathbb{R}^{d'} \to \mathbb{R}^d$ is update function, $\phi : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^{d_e} \to \mathbb{R}^{d'}$ is message function where d_e is the edge feature dimension, $\tau : \mathcal{M} \to \mathbb{R}^d$ is permutation invariant aggregation function and $\mathcal{N}(i)$ is the neighbors of node i in G. Update/message/aggregation functions are usually parametrized by neural networks. For graphs of different types of edges and nodes, one can further extend MPNN to the heterogeneous setting. We use $1, ..., n$ to index graph nodes and vn to denote the virtual node.

Definition 3.3 (heterogeneous MPNN + VN layer)**.** The heterogeneous MPNN + VN layer operates on two types of nodes: 1) virtual node and 2) graph nodes, denoted as vn and gn, and three types of edges: 1) vn-gn edge and 2) gn-gn edges and 3) gn-vn edges. It has the following form

$$\boldsymbol{x}_{\text{vn}}^{(k)} = \gamma_{\text{vn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right) \tag{1}$$

for the virtual node, and

$$\begin{aligned}\boldsymbol{x}_i^{(k)} = \gamma_{\text{gn}}^{(k)}(\boldsymbol{x}_i^{(k-1)}, &\tau_{j\in\mathcal{N}_1(i)}\phi_{\text{gn-vn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right) \\ &+ \tau_{j\in\mathcal{N}_2(i)}\phi_{\text{gn-gn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\Big)\end{aligned} \tag{2}$$

for graph node. Here $\mathcal{N}_1(i)$ for graph node i is the virtual node and $\mathcal{N}_2(i)$ is the set of neighboring graph nodes.

Our proof of approximating self-attention layer $\boldsymbol{L}$ with MPNN layers does not use the graph topology. Next, we introduce a simplified heterogeneous MPNN + VN layer, which will be used in the proof. It is easy to see that setting $\phi_{\text{gn-gn}}^{(k)}$ to be 0 in Definition 3.3 recovers the simplified heterogeneous MPNN + VN layer.

Definition 3.4 (simplified heterogeneous MPNN + VN layer)**.** A simplified heterogeneous MPNN + VN layer is the same as a heterogeneous MPNN + VN layer in Definition 3.3 except we set $\theta_{\text{gn-gn}}$ to be 0. I.e., we have

$$\boldsymbol{x}_{\text{vn}}^{(k)} = \gamma_{\text{vn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right)$$

for the virtual node, and

$$\boldsymbol{x}_i^{(k)} = \gamma_{\text{gn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \tau_{j\in\mathcal{N}_1(i)}\phi_{\text{gn-vn}}^{(k)}\left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i}\right)\right)$$

for graph nodes.

Intuitively, adding the virtual node (VN) to MPNN makes it easy to compute certain quantities, for example, the mean of node features (which is hard for standard MPNN unless the depth is proportional to the diameter of the graph). Using VN thus makes it easy to implement for example the mean subtraction, which helps reduce over-smoothing and improves the performance of GNN (Yang et al., 2020; Zhao & Akoglu, 2019). See more connection between MPNN + VN and over-smoothing in Appendix D.6.

3.2. Assumptions

We have two mild assumptions on feature space $\mathcal{X} \subset \mathbb{R}^{n\times d}$ and the regularity of target function $\boldsymbol{L}$.

AS1. $\forall i \in [n], \boldsymbol{x}_i \in \mathcal{X}_i, \|\boldsymbol{x}_i\| < C_1$. This implies $\mathcal{X}$ is compact.

AS2. $\|\boldsymbol{W}_Q\| < C_2, \|\boldsymbol{W}_K\| < C_2, \|\boldsymbol{W}_V\| < C_2$ for target layer $\boldsymbol{L}$. Combined with AS1 on $\mathcal{X}$, this means $\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is both upper and lower bounded, which further implies $\sum_j e^{\alpha'(\boldsymbol{x}_i,\boldsymbol{x}_j)}$ be both upper bounded and lower bounded.

4. $\mathcal{O}(1)$-depth $\mathcal{O}(1)$-width MPNN + VN for unbiased approximation of attention

The standard self-attention takes $\mathcal{O}(n^2)$ computational time, therefore not scalable for large graphs. Reducing the computational complexity of self-attention in Transformer is active research (Tay et al., 2020). In this section, we consider self-attention in a specific type of efficient transformers, Performer (Choromanski et al., 2020) and Linear Transformer (Katharopoulos et al., 2020b).

One full self-attention layer $\boldsymbol{L}$ is of the following form

$$\boldsymbol{x}_i^{(l+1)} = \sum_{j=1}^{n} \frac{\kappa\left(\boldsymbol{W}_Q^{(l)}\boldsymbol{x}_i^{(l)}, \boldsymbol{W}_K^{(l)}\boldsymbol{x}_j^{(l)}\right)}{\sum_{k=1}^{n}\kappa\left(\boldsymbol{W}_Q^{(l)}\boldsymbol{x}_i^{(l)}, \boldsymbol{W}_K^{(l)}\boldsymbol{x}_k^{(l)}\right)} \cdot \left(\boldsymbol{W}_V^{(l)}\boldsymbol{x}_j^{(l)}\right) \tag{3}$$

where $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the softmax kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) := \exp(\boldsymbol{x}^T\boldsymbol{y})$. The kernel function can be approximated via $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \langle\Phi(\boldsymbol{x}), \Phi(\boldsymbol{y})\rangle_{\mathcal{V}} \approx \phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$ where the first equation is by Mercer's theorem and $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^m$ is a low-dimensional feature map with random transformation. For Performer (Choromanski et al., 2020), the choice of ϕ is taken as $\phi(\boldsymbol{x}) = \frac{\exp\left(\frac{-\|\boldsymbol{x}\|_2^2}{2}\right)}{\sqrt{m}}\left[\exp\left(\boldsymbol{w}_1^T\boldsymbol{x}\right), \cdots, \exp\left(\boldsymbol{w}_m^T\boldsymbol{x}\right)\right]$ where $\boldsymbol{w}_k \sim \mathcal{N}(0, I_d)$ is i.i.d sampled random variable. For Linear Transformer (Katharopoulos et al., 2020b), $\phi(\boldsymbol{x}) = \mathrm{elu}(\boldsymbol{x}) + 1$.

By switching $\kappa(\boldsymbol{x}, \boldsymbol{y})$ to be $\phi(\boldsymbol{x})^T\phi(\boldsymbol{y})$, and denote $\boldsymbol{q}_i = \boldsymbol{W}_Q^{(l)}\boldsymbol{x}_i^{(l)}, \boldsymbol{k}_i = \boldsymbol{W}_K^{(l)}\boldsymbol{x}_i^{(l)}$ and $\boldsymbol{v}_i = \boldsymbol{W}_V^{(l)}\boldsymbol{x}_i^{(l)}$, the approximated version of Equation (3) by Performer and Linear Transformer becomes

$$\begin{aligned}\boldsymbol{x}_i^{(l+1)} &= \sum_{j=1}^{n} \frac{\phi\left(\boldsymbol{q}_i\right)^T\phi\left(\boldsymbol{k}_j\right)}{\sum_{k=1}^{n}\phi\left(\boldsymbol{q}_i\right)^T\phi\left(\boldsymbol{k}_k\right)} \cdot \boldsymbol{v}_j \\ &= \frac{\left(\phi\left(\boldsymbol{q}_i\right)^T\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)\otimes\boldsymbol{v}_j\right)^T}{\phi\left(\boldsymbol{q}_i\right)^T\sum_{k=1}^{n}\phi\left(\boldsymbol{k}_k\right)}.\end{aligned} \tag{4}$$

where we use the matrix multiplication association rule to derive the second equality.

The key advantage of Equation (4) is that $\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)$ and $\sum_{j=1}^{n}\phi(\boldsymbol{k}_j)\otimes\boldsymbol{v}_j$ can be approximated by the virtual node, and shared for all graph nodes, using only $\mathcal{O}(1)$ layers of MPNNs. We denote the self-attention layer of this form in Equation (4) as $\boldsymbol{L}_{\text{Performer}}$. Linear Transformer differs from Performer by choosing a different form of $\phi(\boldsymbol{x}) = \mathrm{Relu}(\boldsymbol{x}) + 1$ in its self-attention layer $\boldsymbol{L}_{\text{Linear-Transformer}}$.

In particular, the VN will approximate $\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)$ and $\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)\otimes\boldsymbol{v}_j$, and represent it as its feature. Both $\phi\left(\boldsymbol{k}_j\right)$ and $\phi\left(\boldsymbol{k}_j\right)\otimes\boldsymbol{v}_j$ can be approximated arbitrarily well by an MLP with constant width (constant in n but can be exponential in d) and depth. Note that $\phi(\boldsymbol{k}_j)\otimes\boldsymbol{v}_j \in \mathbb{R}^{dm}$ but can be reshaped to 1 dimensional feature vector.

More specifically, the initial feature for the virtual node is $\mathbf{1}_{(d+1)m}$, where d is the dimension of node features and m is the number of random projections ω_i. Message function + aggregation function for virtual node $\tau\phi_{\text{vn-gn}} : \mathbb{R}^{(d+1)m} \times \mathcal{M} \to \mathbb{R}^{(d+1)m}$ is

$$\begin{aligned}\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)}(\cdot, \{\boldsymbol{x}_i\}_i) = [&\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right), \\ &\texttt{ReshapeTo1D}(\sum_{j=1}^{n}\phi\left(\boldsymbol{k}_j\right)\otimes\boldsymbol{v}_j)]\end{aligned} \tag{5}$$

where $\texttt{ReshapeTo1D}(\cdot)$ flattens a 2D matrix to a 1D vector in raster order. This function can be arbitrarily approximated by MLP. Note that the virtual node's feature dimension is $(d+1)m$ (where recall m is the dimension of the feature map ϕ used in the linear transformer/Performer), which is larger than the dimension of the graph node d. This is consistent with the early intuition that the virtual node might be overloaded when passing information among nodes. The update function for virtual node $\gamma_{\text{vn}} : \mathbb{R}^{(d+1)m} \times \mathbb{R}^{(d+1)m} \to \mathbb{R}^{(d+1)m}$ is just coping the second argument, which can be exactly implemented by MLP.

VN then sends its message back to all other nodes, where each graph node i applies the update function γ_{gn} :

$\mathbb{R}^{(d+1)m} \times \mathbb{R}^d \to \mathbb{R}^d$ of the form

$$\gamma_{\text{gn}}(\boldsymbol{x}_i, [\sum_{j=1}^{n} \phi(\boldsymbol{k}_j), \texttt{ReshapeTo1D}(\sum_{j=1}^{n} \phi(\boldsymbol{k}_j) \otimes \boldsymbol{v}_j)]) = \frac{\left(\phi(\boldsymbol{q}_i) \sum_{j=1}^{n} \phi(\boldsymbol{k}_j) \otimes \boldsymbol{v}_j\right)^T}{\phi(\boldsymbol{q}_i)^T \sum_{k=1}^{n} \phi(\boldsymbol{k}_k)} \tag{6}$$

to update the graph node feature.

As the update function γ_{gn} can not be computed exactly in MLP, what is left is to show that error induced by using MLP to approximate $\tau\phi_{\text{vn-gn}}$ and γ_{gn} in Equation (5) and Equation (6) can be made arbitrarily small.

Theorem 4.1. *Under the AS1 and AS2, MPNN + VN of $\mathcal{O}(1)$ width and $\mathcal{O}(1)$ depth can approximate $\boldsymbol{L}_{\text{Performer}}$ and $\boldsymbol{L}_{\text{Linear-Transformer}}$ arbitrarily well.*

Proof. We first prove the case of $\boldsymbol{L}_{\text{Performer}}$. We can decompose our target function as the composition of $\tau_{j\in[n]}\phi^{(k)}_{\text{vn-gn}}$, γ_{gn} and ϕ. By the uniform continuity of the functions, it suffices to show that 1) we can approximate ϕ, 2) we can approximate operations in γ_{gn} and $\tau\phi_{\text{vn-gn}}$ arbitrarily well on the compact domain, and 3) the denominator $\phi(\boldsymbol{q}_i)^T \sum_{k=1}^{n} \phi(\boldsymbol{k}_k)$ is uniformly lower bounded by a positive number for any node features in $\mathcal{X}$.

For 1), each component of ϕ is continuous and all inputs $\boldsymbol{k}_j, \mathbf{q}_j$ lie in the compact domain so ϕ can be approximated arbitrarily well by MLP with $\mathcal{O}(1)$ width and $\mathcal{O}(1)$ depth (Cybenko, 1989).

For 2), we need to approximate the operations in γ_{gn} and $\tau\phi_{\text{vn-gn}}$, i.e., approximate multiplication, and vector-scalar division arbitrarily well. As all those operations are continuous, it boils down to showing that all operands lie in a compact domain. By assumption AS1 and AS2 on $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$ and input feature $\mathcal{X}$, we know that $\boldsymbol{q}_i, \boldsymbol{k}_i, \boldsymbol{v}_i$ lies in a compact domain for all graph nodes i. As ϕ is continuous, this implies that $\phi(\boldsymbol{q}_i), \sum_{j=1}^{n} \phi(\boldsymbol{k}_j) \otimes \boldsymbol{v}_j$ lies in a compact domain (n is fixed), therefore the numerator lies in a compact domain. Lastly, since all operations do not involve n, the depth and width are constant in n.

For 3), it is easy to see that $\phi(\boldsymbol{q}_i)^T \sum_{k=1}^{n} \phi(\boldsymbol{k}_k)$ is always positive. We just need to show that the denominator is bound from below by a positive constant. For Performer, $\phi(\boldsymbol{x}) = \frac{\exp\left(\frac{-\|\boldsymbol{x}\|_2^2}{2}\right)}{\sqrt{m}} \left[\exp\left(\boldsymbol{w}_1^T \boldsymbol{x}\right), \cdots, \exp\left(\boldsymbol{w}_m^T \boldsymbol{x}\right)\right]$ where $\boldsymbol{w}_k \sim \mathcal{N}(0, I_d)$. As all norm of input $\boldsymbol{x}$ to ϕ is upper bounded by AS1, $\exp(\frac{-\|\boldsymbol{x}\|_2^2}{2})$ is lower bounded. As m is fixed, we know that $\|\boldsymbol{w}_i^T \boldsymbol{x}\| \leq \|\boldsymbol{w}_i\| \|\boldsymbol{x}\|$, which implies that $\boldsymbol{w}_i^T \boldsymbol{x}$ is lower bounded by $-\|\boldsymbol{w}_i\| \|\boldsymbol{x}\|$ which further implies that $\exp(\boldsymbol{w}_i^T \boldsymbol{x})$ is lower bounded. This means that $\phi(\boldsymbol{q}_i)^T \sum_{k=1}^{n} \phi(\boldsymbol{k}_k)$ is lower bounded.

For Linear Transformer, the proof is essentially the same as above. We only need to show that $\phi(\boldsymbol{x}) = \text{elu}(\boldsymbol{x}) + 1$ is continuous and positive, which is indeed the case. □

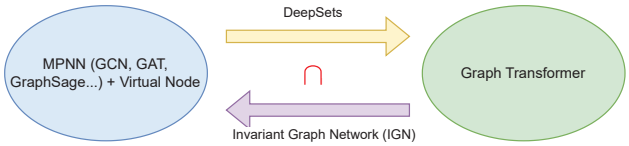

Figure 2: The link between MPNN and GT is drawn via DeepSets in Section 5 of our paper and Invariant Graph Network (IGN) in Kim et al. (2022). Interestingly, IGN is a generalization of DeepSets (Maron et al., 2018).

Besides Performers, there are many other different ways of obtaining linear complexity. In Appendix C.2, we discuss the limitation of MPNN + VN on approximating other types of efficient transformers such as Linformer (Wang et al., 2020b) and Sparse Transformer (Child et al., 2019).

5. $\mathcal{O}(1)$ depth $\mathcal{O}(n^d)$ width MPNN + VN

We have shown that the MPNN + VN can approximate self-attention in Performer and Linear Transformer using only $\mathcal{O}(1)$ depth and $\mathcal{O}(1)$ width. One may naturally wonder whether MPNN + VN can approximate the self-attention layer in the *full* transformer. In this section, we show that MPNN + VN with $O(1)$ depth (number of layers), but with $\mathcal{O}(n^d)$ width, can approximate 1 self-attention layer (and full transformer) arbitrarily well.

The main observation is that MPNN + VN is able to exactly simulate (not just approximate) equivariant DeepSets (Zaheer et al., 2017), which is proved to be universal in approximating any permutation invariant/equivariant maps (Zaheer et al., 2017; Segol & Lipman, 2019). Since the self-attention layer is permutation equivariant, this implies that MPNN + VN can approximate the self-attention layer (and full transformer) with $\mathcal{O}(1)$ depth and $\mathcal{O}(n^d)$ width following a result on DeepSets from Segol & Lipman (2019).

We first introduce the permutation equivariant map, equivariant DeepSets, and permutation equivariant universality.

Definition 5.1 (permutation equivariant map)**.** A map $\boldsymbol{F} : \mathbb{R}^{n\times k} \to \mathbb{R}^{n\times l}$ satisfying $\boldsymbol{F}(\sigma \cdot \boldsymbol{X}) = \sigma \cdot \boldsymbol{F}(\boldsymbol{X})$ for all $\sigma \in S_n$ and $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ is called permutation equivariant.

Definition 5.2 (equivariant DeepSets of Zaheer et al. (2017))**.** Equivariant DeepSets has the following form $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{L}_m^{\text{ds}} \circ \nu \circ \cdots \circ \nu \circ \boldsymbol{L}_1^{\text{ds}}(\boldsymbol{X})$, where $\boldsymbol{L}_i^{\text{ds}}$ is a linear permutation equivariant layer and ν is a nonlinear layer such as ReLU. The linear permutation equivariant layer in DeepSets has the following form $\boldsymbol{L}_i^{\text{ds}}(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{A} + \frac{1}{n}\mathbf{1}\mathbf{1}^T\boldsymbol{X}\boldsymbol{B} + \mathbf{1}\boldsymbol{c}^T$,

where $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d_i \times d_{i+1}}$, $\boldsymbol{c} \in \mathbb{R}^{d_{i+1}}$ is the weights and bias in layer i, and ν is ReLU.

Definition 5.3 (permutation equivariant universality)**.** Given a compact domain $\mathcal{X}$ of $\mathbb{R}^{n \times d_{\text{in}}}$, permutation equivariant universality of a model $\boldsymbol{F} : \mathbb{R}^{n \times d_{\text{in}}} \to \mathbb{R}^{n \times d_{\text{out}}}$ means that for every permutation equivariant continuous function $\boldsymbol{H} : \mathbb{R}^{n \times d_{\text{in}}} \to \mathbb{R}^{n \times d_{\text{out}}}$ defined over $\mathcal{X}$, and any $\epsilon > 0$, there exists a choice of m (i.e., network depth), d_i (i.e., network width at layer i) and the trainable parameters of $\boldsymbol{F}$ so that $\|\boldsymbol{H}(\boldsymbol{X}) - \boldsymbol{F}(\boldsymbol{X})\|_\infty < \epsilon$ for all $\boldsymbol{X} \in \mathcal{X}$.

The universality of equivariant DeepSets is stated as follows.

Theorem 5.4 (Segol & Lipman (2019))**.** *DeepSets with constant layer is universal. Using ReLU activation the width $\omega := max_i d_i$ (d_i is the width for i-th layer of DeepSets) required for universal permutation equivariant network satisfies $\omega \leq d_{out} + d_{in} + \begin{pmatrix} n + d_{in} \\ d_{in} \end{pmatrix} = \mathcal{O}(n^{d_{in}})$.*

We are now ready to state our main theorem.

Theorem 5.5. *MPNN + VN can simulate (not just approximate) equivariant DeepSets: $\mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$. The depth and width of MPNN + VN needed to simulate DeepSets is up to a constant factor of the depth and width of DeepSets. This implies that MPNN + VN of $\mathcal{O}(1)$ depth and $\mathcal{O}(n^d)$ width is permutation equivariant universal, and can approximate self-attention layer and transformers arbitrarily well.*

Proof. Equivariant DeepSets has the following form $\boldsymbol{F}(\boldsymbol{X}) = \boldsymbol{L}_m^{\text{ds}} \circ \nu \circ \cdots \circ \nu \circ \boldsymbol{L}_1^{\text{ds}}(\boldsymbol{X})$, where $\boldsymbol{L}_i^{\text{ds}}$ is the linear permutation equivariant layer and ν is an entrywise nonlinear activation layer. Recall that the linear equivariant layer has the form $\boldsymbol{L}_i^{\text{ds}}(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{A} + \frac{1}{n}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X}\boldsymbol{B} + \boldsymbol{1}\boldsymbol{c}^T$. As one can use the same nonlinear entrywise activation layer ν in MPNN + VN, it suffices to prove that MPNN + VN can compute linear permutation equivariant layer $\boldsymbol{L}^{\text{ds}}$. Now we show that 2 layers of MPNN + VN can exactly simulate any given linear permutation equivariant layer $\boldsymbol{L}^{\text{ds}}$.

Specifically, at layer 0, we initialized the node features as follows: The VN node feature is set to 0, while the node feature for the i-th graph node is set up as $\boldsymbol{x}_i \in \mathbb{R}^d$.

At layer 1: VN node feature is $\frac{1}{n}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X}$, average of node features. The collection of features over n graph node feature is $\boldsymbol{X}\boldsymbol{A}$. We only need to transform graph node features by a linear transformation, and set the VN feature as the average of graph node features in the last iteration. Both can be exactly implemented in Definition 3.4 of simplified heterogeneous MPNN + VN.

At layer 2: VN node feature is set to be 0, and the graph node feature is $\boldsymbol{X}\boldsymbol{A} + \frac{1}{n}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X}\boldsymbol{B} + \boldsymbol{1}\boldsymbol{c}^T$. Here we only need to perform the matrix multiplication of the VN feature with $\boldsymbol{B}$, as well as add a bias $\boldsymbol{c}$. This can be done by implementing a linear function for γ_{gn}.

It is easy to see the width required for MPNN + VN to simulate DeepSets is constant. Thus, one can use 2 layers of MPNN + VN to compute linear permutation equivariant layer $\boldsymbol{L}_i^{\text{ds}}$, which implies that MPNN + VN can simulate 1 layer of DeepSets exactly with constant depth and constant width (independent of n). Then by the universality of DeepSets, stated in Theorem 5.4, we conclude that MPNN + VN is also permutation equivariant universal, which implies that the constant layer of MPNN + VN with $\mathcal{O}(n^d)$ width is able to approximate any continuous equivariant maps. As the self-attention layer $\boldsymbol{L}$ and full transformer are both continuous and equivariant, they can be approximated by MPNN + VN arbitrarily well. □

Thanks to the connection between MPNN + VN with DeepSets, there is no extra assumption on $\mathcal{X}$ except for being compact. The drawback on the other hand is that the upper bound on the computational complexity needed to approximate the self-attention with wide MPNN + VN is worse than directly computing self-attention when $d > 2$.

6. $\mathcal{O}(n)$ depth $\mathcal{O}(1)$ width MPNN + VN

The previous section shows that we can approximate a full attention layer in Transformer using MPNN with $\mathcal{O}(1)$ depth but $\mathcal{O}(n^d)$ width where n is the number of nodes and d is the dimension of node features. In practice, it is not desirable to have the width depend on the graph size.

In this section, we hope to study MPNN + VNs with $\mathcal{O}(1)$ width and their ability to approximate a self-attention layer in the Transformer. However, this appears to be much more challenging. Our result in this section only shows that for a rather restrictive family of input graphs (see Assumption 3 below), we can approximate a full self-attention layer of transformer with an MPNN + VN of $\mathcal{O}(1)$ width and $\mathcal{O}(n)$ depth. We leave the question of MPNN + VN's ability in approximate transformers for more general families of graphs for future investigation.

We first introduce the notion of $(\boldsymbol{V}, \delta)$ separable node features. This is needed to ensure that VN can approximately select one node feature to process at each iteration with attention α_{vn}, the self-attention in the virtual node.

Definition 6.1 ($(\boldsymbol{V}, \delta)$ separable by α)**.** Given a graph G of size n and a fixed $\boldsymbol{V} \in \mathbb{R}^{n \times d} = [\boldsymbol{v}_1, ..., \boldsymbol{v}_n]$ and $\bar{\alpha} \in \mathcal{A}$, we say node feature $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ of G is $(\boldsymbol{V}, \delta)$ separable by some $\bar{\alpha}$ if the following holds. For any node feature $\boldsymbol{x}_i$, there exist weights $\boldsymbol{W}_K^{\bar{\alpha}}, \boldsymbol{W}_Q^{\bar{\alpha}}$ in attention score $\bar{\alpha}$ such that $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j \neq i} \bar{\alpha}(\boldsymbol{x}_j, \boldsymbol{v}_i) + \delta$. We say set $\mathcal{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$ if every element $\boldsymbol{X} \in \mathcal{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$.

Table 2: Baselines for `Peptides-func` (graph classification) and `Peptides-struct` (graph regression). The performance metric is Average Precision (AP) for classification and MAE for regression. **Bold**: Best score.

Model	# Params.	Peptides-func		Peptides-struct	
		Test AP before VN	Test AP after VN ↑	Test MAE before VN	Test MAE after VN ↓
GCN	508k	0.5930±0.0023	0.6623±0.0038	0.3496±0.0013	**0.2488±0.0021**
GINE	476k	0.5498±0.0079	0.6346±0.0071	0.3547±0.0045	0.2584±0.0011
GatedGCN	509k	0.5864±0.0077	0.6635±0.0024	0.3420±0.0013	0.2523±0.0016
GatedGCN+RWSE	506k	0.6069±0.0035	**0.6685±0.0062**	0.3357±0.0006	0.2529±0.0009
Transformer+LapPE	488k	0.6326±0.0126	-	0.2529±0.0016	-
SAN+LapPE	493k	0.6384±0.0121	-	0.2683±0.0043	-
SAN+RWSE	500k	0.6439±0.0075	-	0.2545±0.0012	-

Table 3: Test performance in graph-level OGB benchmarks (Hu et al., 2020). Shown is the mean ± s.d. of 10 runs.

Model	ogbg-molhiv	ogbg-molpcba	ogbg-ppa	ogbg-code2
	AUROC ↑	Avg. Precision ↑	Accuracy ↑	F1 score ↑
GCN	0.7606 ± 0.0097	0.2020 ± 0.0024	0.6839 ± 0.0084	0.1507 ± 0.0018
GCN+virtual node	0.7599 ± 0.0119	0.2424 ± 0.0034	0.6857 ± 0.0061	0.1595 ± 0.0018
GIN	0.7558 ± 0.0140	0.2266 ± 0.0028	0.6892 ± 0.0100	0.1495 ± 0.0023
GIN+virtual node	0.7707 ± 0.0149	0.2703 ± 0.0023	0.7037 ± 0.0107	0.1581 ± 0.0026
SAN	0.7785 ± 0.2470	0.2765 ± 0.0042	–	–
GraphTrans (GCN-Virtual)	–	0.2761 ± 0.0029	–	0.1830 ± 0.0024
K-Subtree SAT	–	–	0.7522 ± 0.0056	0.1937 ± 0.0028
GPS	0.7880 ± 0.0101	0.2907 ± 0.0028	0.8015 ± 0.0033	0.1894 ± 0.0024
MPNN + VN + NoPE	0.7676 ± 0.0172	0.2823 ± 0.0026	0.8055 ± 0.0038	0.1727 ± 0.0017
MPNN + VN + PE	0.7687 ± 0.0136	0.2848 ± 0.0026	0.8027 ± 0.0026	0.1719 ± 0.0013

The use of $(\boldsymbol{V}, \delta)$ separability is to approximate hard selection function arbitrarily well, which is stated below and proved in Appendix B.1.

Lemma 6.2 (approximate hard selection)**.** *Given $\mathcal{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$ for some fixed $\boldsymbol{V} \in \mathbb{R}^{n\times d}$, $\bar{\alpha} \in \mathcal{A}$ and $\delta > 0$, the following holds. For any $\epsilon > 0$ and $i \in [n]$, there exists a set of attention weights $\boldsymbol{W}_{i,Q}, \boldsymbol{W}_{i,K}$ in i-th layer of MPNN + VN such that $\alpha_{vn}(\boldsymbol{x}_i, \boldsymbol{v}_i) > 1 - \epsilon$ for any $\boldsymbol{x}_i \in \mathcal{X}_i$. In other words, we can approximate a hard selection function $f_i(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = \boldsymbol{x}_i$ arbitrarily well on $\mathcal{X}$ by setting $\alpha_{vn} = \bar{\alpha}$.*

With the notation set up, We now state an extra assumption needed for deep MPNN + VN case and the main theorem.

AS3. $\mathcal{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$ for some fixed $\boldsymbol{V} \in \mathbb{R}^{n\times d}$, $\bar{\alpha} \in \mathcal{A}$ and $\delta > 0$.

Theorem 6.3. *Assume AS 1-3 hold for the compact set $\mathcal{X}$ and $\boldsymbol{L}$. Given any graph G of size n with node features $\boldsymbol{X} \in \mathcal{X}$, and a self-attention layer $\boldsymbol{L}$ on G (fix $\boldsymbol{W}_K, \boldsymbol{W}_Q, \boldsymbol{W}_V$ in α), there exists a $\mathcal{O}(n)$ layer of heterogeneous MPNN + VN with the specific aggregate/update/message function that can approximate $\boldsymbol{L}$ on $\mathcal{X}$ arbitrarily well.*

The proof is presented in the Appendix B. On the high level, we can design an MPNN + VN where the i-th layer will select $\tilde{\boldsymbol{x}}_i$, an approximation of $\boldsymbol{x}_i$ via attention mechanism, enabled by Lemma 6.2, and send $\tilde{\boldsymbol{x}}_i$ to the virtual node. Virtual node will then pass the $\tilde{\boldsymbol{x}}_i$ to all graph nodes and computes the approximation of $e^{\alpha(\boldsymbol{x}_i, \boldsymbol{x}_j)}, \forall j \in [n]$. Repeat such procedures n times for all graph nodes, and finally, use the last layer for attention normalization. A slight relaxation of AS3 is also provided in the appendix.

7. Experiments

We benchmark MPNN + VN for three tasks, long range interaction modeling in Section 7.1, OGB regression tasks in Section 7.2, and focasting sea surface temperature in Section 7.3. The code is available `https://github.com/Chen-Cai-OSU/MPNN-GT-Connection`.

7.1. MPNN + VN for LRGB Datasets

We experiment with MPNN + VN for Long Range Graph Benchmark (LRGB) datasets. Original paper (Dwivedi et al., 2022) observes that GT outperforms MPNN on 4 out of 5 datasets, among which GT shows significant improvement over MPNN on `Peptides-func` and `Peptides-struct` for all MPNNs. To test the effectiveness of the virtual node, we take the original code and modify the graph topology by adding a virtual node and keeping the hyperparameters of all models unchanged.

Results are in Table 2. Interestingly, such a simple change can boost MPNN + VN by a large margin on

Table 4: Evaluation on PCQM4Mv2 (Hu et al., 2021) dataset. For GPS evaluation, we treated the *validation* set of the dataset as a test set, since the *test-dev* set labels are private.

Model	PCQM4Mv2			
	Test-dev MAE ↓	Validation MAE ↓	Training MAE	# Param.
GCN	0.1398	0.1379	n/a	2.0M
GCN-virtual	0.1152	0.1153	n/a	4.9M
GIN	0.1218	0.1195	n/a	3.8M
GIN-virtual	0.1084	0.1083	n/a	6.7M
GRPE (Park et al., 2022)	0.0898	0.0890	n/a	46.2M
EGT (Hussain et al., 2022)	0.0872	0.0869	n/a	89.3M
Graphormer (Shi et al., 2022)	n/a	0.0864	0.0348	48.3M
GPS-small	n/a	0.0938	0.0653	6.2M
GPS-medium	n/a	0.0858	0.0726	19.4M
MPNN + VN + PE (small)	n/a	0.0942	0.0617	5.2M
MPNN + VN + PE (medium)	n/a	0.0867	0.0703	16.4M
MPNN + VN + NoPE (small)	n/a	0.0967	0.0576	5.2M
MPNN + VN + NoPE (medium)	n/a	0.0889	0.0693	16.4M

`Peptides-func` and `Peptides-struct`. Notably, with the addition of VN, GatedGCN + RWSE (random-walk structural encoding) after augmented by VN **outperforms all transformers** on `Peptides-func`, and GCN outperforms transformers on `Peptides-struct`.

7.2. Stronger MPNN + VN Implementation

Next, by leveraging the modularized implementation from GraphGPS (Rampášek et al., 2022), we implemented a version of MPNN + VN with/without extra positional embedding. Our goal is not to achieve SOTA but instead to push the limit of MPNN + VN and better understand the source of the performance gain for GT. In particular, we replace the GlobalAttention Module in GraphGPS with DeepSets, which is equivalent to one specific version of MPNN + VN. We tested this specific version of MPNN + VN on 4 OGB datasets, both with and without the use of positional embedding. The results are reported in Table 3. Interestingly, even without the extra position embedding, our MPNN + VN is able to further improve over the previous GCN + VN & GIN + VN implementation. The improvement on **ogbg-ppa** is particularly impressive, which is from 0.7037 to 0.8055. Furthermore, it is important to note that while MPNN + VN does not necessarily outperform GraphGPS, which is a state-of-the-art architecture using both MPNN, Position/structure encoding and Transformer, the difference is quite small – this however, is achieved by a simple MPNN + VN architecture.

We also test MPNN + VN on large-scale molecule datasets PCQMv2, which has 529,434 molecule graphs. We followed (Rampášek et al., 2022) and used the original validation set as the test set, while we left out random 150K molecules for our validation set. As we can see from Table 4, MPNN + VN + NoPE performs significantly better than the early MPNN + VN implementation: GIN + VN and GCN + VN. The performance gap between GPS on the other hand is rather small: 0.0938 (GPS) vs. 0.0942 (MPNN + VN + PE) for the small model and 0.0858 (GPS) vs. 0.0867 (MPNN + VN + PE) for the medium model.

7.3. Forecasting Sea Surface Temperature

In this experiment, we apply our MPNN + VN model to forecast sea surface temperature (SST). We are particularly interested in the empirical comparison between MPNN + VN and Linear Transformer (Katharopoulos et al., 2020a) as according to Section 4, MPNN + VN theoretically can approximate Linear Transformer.

In particular, from the DOISST data proposed by (Huang et al., 2021), we construct a dataset of daily SST in the Pacific Ocean from 1982 to 2021, in the region of longitudes from $180.125°$E to $269.875°$E and latitudes from $-14.875°$N to $14.875°$N. Following the procedure from (de Bezenac et al., 2018; de Bézenac et al., 2019) and Wang et al. (2022), we divide the region into 11 batches of equal size with 30 longitudes and 30 latitudes at $0.5°$-degree resolution, that can be represented as a graph of 900 nodes. The tasks are to predict the next 4 weeks, 2 weeks and 1 week of SST at each location, given 6 weeks of historical data. We train on data from years 1982–2018, validate on data from 2019 and test on data from 2020–2021. The number of training, validation, and testing examples are roughly 150K, 3K, and 7K. See details of dataset construction, model architectures, and training scheme in Appendix D.6.

We compare our model to other baselines including TF-Net (Wang et al., 2020a), a SOTA method for spatiotemporal forecasting, Linear Transformer (Katharopoulos et al., 2020a; Wang et al., 2020b) with Laplacian positional encoding (LapPE), and Multilayer Perceptron (MLP). We use Mean Square Error (MSE) as the metric and report the errors on the test set, shown in the Table 5. We observe that the virtual node (VN) alone improves upon MPNN by 3.8%, 6.6% and 4.5% in 4-, 2- and 1-week settings, respectively.

Table 5: Results of SST prediction.

Model	4 weeks	2 weeks	1 week
MLP	0.3302	0.2710	0.2121
TF-Net	0.2833	**0.2036**	**0.1462**
Linear Transformer + LapPE	0.2818	0.2191	0.1610
MPNN	0.2917	0.2281	0.1613
MPNN + VN	**0.2806**	0.2130	0.1540

Furthermore, aligned with our theory in Section 4, MPNN + VN indeed achieves comparable results with Linear Transformer and outperforms it by a margin of 0.4%, 2.8% and 4.3% in 4-, 2- and 1-week settings, respectively.

8. Concluding Remarks

In this paper, we study the expressive power of MPNN + VN under the lens of GT. If we target the self-attention layer in Performer and Linear Transformer, one only needs $\mathcal{O}(1)$-depth $\mathcal{O}(1)$ width for arbitrary approximation error. For self-attention in full transformer, we prove that heterogeneous MPNN + VN of either $\mathcal{O}(1)$ depth $\mathcal{O}(n^d)$ width or $\mathcal{O}(n)$ depth $\mathcal{O}(1)$ width (under some assumptions) can approximate 1 self-attention layer arbitrarily well. Compared to early results (Kim et al., 2022) showing GT can approximate MPNN, our theoretical result draws the connection from the inverse direction.

On the empirical side, we demonstrate that MPNN + VN remains a surprisingly strong baseline. Despite recent efforts, we still lack good benchmark datasets where GT can outperform MPNN by a large margin. Understanding the inductive bias of MPNN and GT remains challenging. For example, can we mathematically characterize tasks that require effective long-range interaction modeling, and provide a theoretical justification for using GT over MPNN (or vice versa) for certain classes of functions on the space of graphs? We believe making processes towards answering such questions is an important future direction for the graph learning community.

Acknowledgement

This work was supported in part by the U.S. Department Of Energy, Office of Science, U. S. Army Research Office under Grant W911NF-20-1-0334, Google Faculty Award, Amazon Research Award, and NSF Grants #2134274, #2107256, #2134178, CCF-2217033, and CCF-2112665.

References

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Brody, S., Alon, U., and Yahav, E. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Cai, C. and Wang, Y. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020.

Chen, D., O'Bray, L., and Borgwardt, K. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pp. 3469–3489. PMLR, 2022.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

de Bezenac, E., Pajot, A., and Gallinari, P. Deep learning for physical processes: Incorporating prior scientific knowledge. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=By4HsfWAZ.

de Bézenac, E., Pajot, A., and Gallinari, P. Deep learning for physical processes: incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, dec 2019. doi: 10.1088/1742-5468/ab3195. URL https://dx.doi.org/10.1088/1742-5468/ab3195.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. *arXiv preprint arXiv:2206.08164*, 2022.

d'Ascoli, S., Touvron, H., Leavitt, M. L., Morcos, A. S., Biroli, G., and Sagun, L. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, pp. 2286–2296. PMLR, 2021.

Geisler, S., Li, Y., Mankowitz, D., Cemgil, A. T., Günnemann, S., and Paduraru, C. Transformers meet directed graphs. *arXiv preprint arXiv:2302.00049*, 2023.

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

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 2022.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

Huang, B., Liu, C., Banzon, V., Freeman, E., Graham, G., Hankins, B., Smith, T., and Zhang, H.-M. Improvements of the daily optimum interpolation sea surface temperature (doisst) version 2.1. *Journal of Climate*, 34(8):2923 – 2939, 2021. doi: 10.1175/JCLI-D-20-0166.1. URL https://journals.ametsoc.org/view/journals/clim/34/8/JCLI-D-20-0166.1.xml.

Hussain, M. S., Zaki, M. J., and Subramanian, D. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 655–665, 2022.

Hwang, E., Thost, V., Dasgupta, S. S., and Ma, T. An analysis of virtual nodes in graph neural networks for link prediction. In *Learning on Graphs Conference*, 2022.

Kalyan, K. S., Rajasekharan, A., and Sangeetha, S. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv preprint arXiv:2108.05542*, 2021.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020a. URL https://arxiv.org/abs/2006.16236.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020b.

Kim, J., Nguyen, T. D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. Pure transformers are powerful graph learners. *arXiv preprint arXiv:2207.02505*, 2022.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI conference on artificial intelligence*, 2018.

Lim, D., Robinson, J., Zhao, L., Smidt, T., Sra, S., Maron, H., and Jegelka, S. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10012–10022, 2021.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

Mialon, G., Chen, D., Selosse, M., and Mairal, J. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

Müller, L., Galkin, M., Morris, C., and Rampášek, L. Attending to graph transformers. *arXiv preprint arXiv:2302.04181*, 2023.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Park, W., Chang, W.-G., Lee, D., Kim, J., et al. Grpe: Relative positional encoding for graph transformer. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.

Reynolds, R. W., Smith, T. M., Liu, C., Chelton, D. B., Casey, K. S., and Schlax, M. G. Daily high-resolution blended analyses for sea surface temperature. *J. Climate*, 20:5473–5496, 2007.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. A simple neural network module for relational reasoning. *Advances in neural information processing systems*, 30, 2017.

Segol, N. and Lipman, Y. On universal equivariant set networks. *arXiv preprint arXiv:1910.02421*, 2019.

Shi, Y., Zheng, S., Ke, G., Shen, Y., You, J., He, J., Luo, S., Liu, C., He, D., and Liu, T.-Y. Benchmarking graphormer on large-scale molecular modeling datasets. *arXiv preprint arXiv:2203.04810*, 2022.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)*, 2020.

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., and Bronstein, M. M. Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Wagstaff, E., Fuchs, F. B., Engelcke, M., Osborne, M. A., and Posner, I. Universal approximation of functions on sets. *Journal of Machine Learning Research*, 23(151): 1–56, 2022.

Wang, R., Kashinath, K., Mustafa, M., Albert, A., and Yu, R. Towards physics-informed deep learning for turbulent flow prediction. pp. 1457–1466, 08 2020a. doi: 10.1145/3394486.3403198.

Wang, R., Walters, R., and Yu, R. Meta-learning dynamics forecasting using task inference. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=BsSP7pZGFQO.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020b.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Wu, Q., Zhao, W., Li, Z., Wipf, D., and Yan, J. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, 2022.

Wu, Z., Jain, P., Wright, M., Mirhoseini, A., Gonzalez, J. E., and Stoica, I. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021.

Yang, C., Wang, R., Yao, S., Liu, S., and Abdelzaher, T. Revisiting over-smoothing in deep gcns. *arXiv preprint arXiv:2003.13663*, 2020.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhao, L. and Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. *arXiv preprint arXiv:1909.12223*, 2019.

Zweig, A. and Bruna, J. Exponential separations in symmetric neural networks. *arXiv preprint arXiv:2206.01266*, 2022.

A. Notations

We provide a notation table for references.

Table 6: Summary of important notations.

Symbol	Meaning
$\boldsymbol{X} \in \mathcal{X} \subset \mathbb{R}^{n \times d}$	graph node features
$\boldsymbol{x}_i \in \mathbb{R}^{1 \times d}$	graph node i's feature
$\tilde{\boldsymbol{x}}_i \in \mathbb{R}^{1 \times d}$	approximated graph node i's feature via attention selection
$\mathcal{M}$	A multiset of vectors in $\mathbb{R}^d$
$\boldsymbol{W}_Q^{(l)}, \boldsymbol{W}_K^{(l)}, \boldsymbol{W}_V^{(l)} \in \mathbb{R}^{d \times d'}$	attention matrix of l-th self-attention layer in graph transformer
$\mathcal{X}$	feature space
$\mathcal{X}_i$	projection of feature space onto i-th coordinate
$\boldsymbol{L}_i^{\text{ds}}$	i-th linear permutation equivariant layer in DeepSets
$\boldsymbol{L}, \boldsymbol{L}'$	full self attention layer; approximate self attention layer in Performer
$\boldsymbol{z}_{\text{vn}}^{(l)}, \boldsymbol{z}_i^{(l)}$	virtual/graph node feature at layer l of heterogeneous MPNN + VN
α_{vn}	attention score in MPNN + VN
$\alpha(\cdot, \cdot)$	normalized attention score
$\alpha_{\text{GATv2}}(\cdot, \cdot)$	normalized attention score with GATv2
$\alpha'(\cdot, \cdot)$	unnormalized attention score. $\alpha'(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}\boldsymbol{W}_Q(\boldsymbol{W}_K)^T\boldsymbol{v}^T$
$\alpha'_{\text{GATv2}}(\cdot, \cdot)$	unnormalized attention score with GATv2. $\alpha'_{\text{GATv2}}(\boldsymbol{u}, \boldsymbol{v}) := \boldsymbol{a}^T \text{LeakyReLU}\left(\boldsymbol{W} \cdot [\boldsymbol{u} \| \boldsymbol{v}] + \boldsymbol{b}\right)$
$\mathcal{A}$	space of attentions, where each element $\alpha \in \mathcal{A}$ is of form $\alpha(\boldsymbol{u}, \boldsymbol{v}) = \text{softmax}(\boldsymbol{u}\boldsymbol{W}_Q(\boldsymbol{W}_K)^T\boldsymbol{v}^T)$
C_1	upper bound on norm of all node features $\|\boldsymbol{x}_i\|$
C_2	upper bound on the norm of $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V$ in target $\boldsymbol{L}$
C_3	upper bound on the norm of attention weights of α_{vn} when selecting $\boldsymbol{x}_i$
$\gamma^{(k)}(\cdot, \cdot)$	update function
$\theta^{(k)}(\cdot, \cdot)$	message function
$\tau(\cdot)$	aggregation function

B. $\mathcal{O}(n)$ Heterogeneous MPNN + VN Layer with $O(1)$ Width Can Approximate 1 Self Attention Layer Arbitrarily Well

B.1. Assumptions

A special case of $(\boldsymbol{V}, \delta)$ separable is when $\delta = 0$, i.e., $\forall i, \bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j \neq i} \bar{\alpha}(\boldsymbol{x}_j, \boldsymbol{v}_i)$. We provide a geometric characterization of $\boldsymbol{X}$ being $(\boldsymbol{V}, 0)$ separable.

Lemma B.1. *Given $\bar{\alpha}$ and $\boldsymbol{V}$, $\boldsymbol{X}$ is $(\boldsymbol{V}, 0)$ separable by $\bar{\alpha} \iff \boldsymbol{x}_i$ is not in the convex hull spanned by $\{\boldsymbol{x}_j\}_{j \neq i}$. $\iff$ there are no points in the convex hull of $\{\boldsymbol{x}_i\}_{i \in [n]}$.*

Proof. The second equivalence is trivial so we only prove the first equivalence. By definition, $\boldsymbol{X}$ is $(\boldsymbol{V}, 0)$ separable by $\bar{\alpha}$ $\iff \bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j \neq i} \bar{\alpha}(\boldsymbol{x}_j, \boldsymbol{v}_i) \forall i \in [n] \iff \langle \boldsymbol{x}_i, \boldsymbol{W}_Q^{\bar{\alpha}} \boldsymbol{W}_K^{\bar{\alpha}, T} \boldsymbol{v}_i \rangle > \max_{j \neq i} \langle \boldsymbol{x}_j, \boldsymbol{W}_Q^{\bar{\alpha}} \boldsymbol{W}_K^{\bar{\alpha}, T} \boldsymbol{v}_i \rangle \forall i \in [n]$.

By denoting the $\boldsymbol{v}'_i := \boldsymbol{W}_Q^{\bar{\alpha}} \boldsymbol{W}_K^{\bar{\alpha}, T} \boldsymbol{v}_i \in \mathbb{R}^d$, we know that $\langle \boldsymbol{x}_i, \boldsymbol{v}'_i \rangle > \max_{j \neq i} \langle \boldsymbol{x}_j, \boldsymbol{v}'_i \rangle \forall i \in [n]$, which implies that $\forall i \in [n], \boldsymbol{x}_i$ can be linearly seprated from $\{\boldsymbol{x}_j\}_{j \neq i} \iff \boldsymbol{x}_i$ is not in the convex hull spanned by $\{\boldsymbol{x}_j\}_{j \neq i}$, which concludes the proof. □

Lemma B.2 (approximate hard selection)**.** *Given $\mathcal{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$ for some fixed $\boldsymbol{V} \in \mathbb{R}^{n \times d}$, $\bar{\alpha} \in \mathcal{A}$ and $\delta > 0$, the following holds. For any $\epsilon > 0$ and $i \in [n]$, there exists a set of attention weights $\boldsymbol{W}_{i,Q}, \boldsymbol{W}_{i,K}$ in i-th layer of MPNN + VN such that $\alpha_{vn}(\boldsymbol{x}_i, \boldsymbol{v}_i) > 1 - \epsilon$ for any $\boldsymbol{x}_i \in \mathcal{X}_i$. In other words, we can approximate a hard selection function $f_i(\boldsymbol{x}_1, ..., \boldsymbol{x}_n) = \boldsymbol{x}_i$ arbitrarily well on $\mathcal{X}$ by setting $\alpha_{vn} = \bar{\alpha}$.*

Proof. Denote $\bar{\alpha}'$ as the unnormalized $\bar{\alpha}$. As $\mathcal{X}$ is $(\boldsymbol{V}, \delta)$ separable by $\bar{\alpha}$, by definition we know that $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j \neq i} \bar{\alpha}(\boldsymbol{x}_j, \boldsymbol{v}_i) + \delta$ holds for any $i \in [n]$ and $\boldsymbol{x}_i \in \mathcal{M}$. We can amplify this by multiple the weight matrix in $\bar{\alpha}$ by a

constant factor c to make $\bar{\alpha}'(\boldsymbol{x}_i, \boldsymbol{v}_i) > \max_{j \neq i} \bar{\alpha}'(\boldsymbol{x}_j, \boldsymbol{v}_i) + c\delta$. This implies that $e^{\bar{\alpha}'(\boldsymbol{x}_i, \boldsymbol{v}_i)} > e^{c\delta} \max_{j \neq i} e^{\bar{\alpha}'(\boldsymbol{x}_j, \boldsymbol{v}_i)}$. This means after softmax, the attention score $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i)$ will be at least $\frac{e^{c\delta}}{e^{c\delta}+n-1}$. We can pick a large enough $c(\delta, \epsilon)$ such that $\bar{\alpha}(\boldsymbol{x}_i, \boldsymbol{v}_i) > 1 - \epsilon$ for any $\boldsymbol{x}_i \in \mathcal{X}_i$ and $\epsilon > 0$. □

Proof Intuition and Outline. On the high level, i-th MPNN + VN layer will select $\tilde{\boldsymbol{x}}_i$, an approximation i-th node feature $\boldsymbol{x}_i$ via attention mechanism, enabled by Lemma 6.2, and send $\tilde{\boldsymbol{x}}_i$ to the virtual node. Virtual node will then pass the $\tilde{\boldsymbol{x}}_i$ to all graph nodes and computes the approximation of $e^{\alpha(\boldsymbol{x}_i, \boldsymbol{x}_j)}, \forall j \in [n]$. Repeat such procedures n times for all graph nodes, and finally, use the last layer for attention normalization.

The main challenge of the proof is to 1) come up with message/update/aggregation functions for heterogeneous MPNN + VN layer, which is shown in Appendix B.2, and 2) ensure the approximation error, both from approximating Aggregate/Message/Update function with MLP and the noisy input, can be well controlled, which is proved in Appendix B.4.

We will first instantiate the Aggregate/Message/Update function for virtual/graph nodes in Appendix B.2, and prove that each component can be either exactly computed or approximated to an arbitrary degree by MLP. Then we go through an example in Appendix B.3 of approximate self-attention layer $\boldsymbol{L}$ with $\mathcal{O}(n)$ MPNN + VN layers. The main proof is presented in Appendix B.4, where we show that the approximation error introduced during different steps is well controlled. Lastly, in Appendix B.5 we show assumption on node features can be relaxed if a more powerful attention mechanism `GATv2` (Brody et al., 2021) is allowed in MPNN + VN.

B.2. Aggregate/Message/Update Functions

Let $\mathcal{M}$ be a multiset of vectors in $\mathbb{R}^d$. The specific form of Aggregate/Message/Update for virtual and graph nodes are listed below. Note that ideal forms will be implemented as MLP, which will incur an approximation error that can be controlled to an arbitrary degree. We use $\boldsymbol{z}_{\text{vn}}^{(k)}$ denotes the virtual node's feature at l-th layer, and $\boldsymbol{z}_i^{(k)}$ denotes the graph node i's node feature. Iteration index k starts with 0 and the node index starts with 1.

B.2.1. VIRTUAL NODE

At k-th iteration, virtual node i's feature $\boldsymbol{z}_i^{(k)}$ is a concatenation of three component $[\tilde{\boldsymbol{x}}_i, \boldsymbol{v}_{k+1}, 0]$ where the first component is the approximately selected node features $\boldsymbol{x}_i \in \mathbb{R}^d$, the second component is the $\boldsymbol{v}_i \in \mathbb{R}^d$ that is used to select the node feature in i-th iteration. The last component is just a placeholder to ensure the dimension of the virtual node and graph node are the same. It is introduced to simplify notation.

Initial feature is $\boldsymbol{z}_{\text{vn}}^{(0)} = [\boldsymbol{0}_d, \boldsymbol{v}_1, 0]$.

Message function + Aggregation function $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)} : \mathbb{R}^{2d+1} \times \mathcal{M} \to \mathbb{R}^{2d+1}$ has two cases to discuss depending on value of k. For $k = 1, 2, ..., n$,

$$\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)}(\boldsymbol{z}_{\text{vn}}^{(k-1)}, \{\boldsymbol{z}_i^{(k-1)}\}_i) = \begin{cases} \sum_i \alpha_{\text{vn}}(\boldsymbol{z}_{\text{vn}}^{(k-1)}, \boldsymbol{z}_i^{(k-1)})\boldsymbol{z}_i^{(k-1)} & k = 1, 2, ..., n \\ \boldsymbol{1}_{2d+1} & k = n+1, n+2 \end{cases} \tag{7}$$

where $\boldsymbol{z}_{\text{vn}}^{(k-1)} = [\tilde{\boldsymbol{x}}_{k-1}, \boldsymbol{v}_k, 0]$. $\boldsymbol{z}_i^{(k-1)} = [\overbrace{\underbrace{\boldsymbol{x}_i}_{d \text{ dim}}, ..., ...}^{2d+1 \text{ dim}}]$ is the node i's feature, where the first d coordinates remain fixed for different iteration k. $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)}$ use attention α_{vn} to approximately select k-th node feature $[\overbrace{\underbrace{\boldsymbol{x}_k}_{d \text{ dim}}, ..., ...}^{2d+1 \text{ dim}}]$. Note that the particular form of attention α_{vn} needed for soft selection is not important as long as we can approximate hard selection arbitrarily well. As the $\boldsymbol{z}_{\text{vn}}^{(k-1)}$ contains $\boldsymbol{v}_k$ and $\boldsymbol{z}_i^{(k-1)}$ contains $\boldsymbol{x}_i$ (see definition of graph node feature in Appendix B.2.2), this step can be made as close to hard selection as possible, according to Lemma B.6.

In the case of $k = n+1$, $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(k)} : \underbrace{\mathbb{R}^{2d+1}}_{\text{vn}} \times \underbrace{\mathcal{M}}_{\text{set of gn}} \to \mathbb{R}^d$ simply returns $\boldsymbol{1}_{2d+1}$. This can be exactly implemented by an MLP.

Update function $\gamma_{vn}^{(k)} : \underbrace{\mathbb{R}^{2d+1}}_{vn} \times \underbrace{\mathbb{R}^{2d+1}}_{gn} \to \mathbb{R}^{2d+1}$: Given the virtual node's feature in the last iteration, and the selected feature in virtual node $\boldsymbol{y} = [\boldsymbol{x}_k, ..., ...]$ with α_{vn},

$$\gamma_{\text{vn}}^{(k)}(\cdot, \boldsymbol{y}) = \begin{cases} [\boldsymbol{y}_{0:d}, \boldsymbol{v}_{k+1}, 0] & k = 1, ..., n-1 \\ [\boldsymbol{y}_{0:d}, \mathbf{0}_d, 0] & k = n \\ \mathbf{1}_{2d+1} & k = n+1, n+2 \end{cases} \tag{8}$$

where $\boldsymbol{y}_{0:d}$ denotes the first d channels of $\boldsymbol{y} \in \mathbb{R}^{2d+1}$. $\boldsymbol{y}$ denotes the selected node $\boldsymbol{z}_i$'s feature in Message/Aggregation function. $\gamma_{\text{vn}}^{(k)}$ can be exactly implemented by an MLP for any $k = 1, ..., n+2$.

B.2.2. GRAPH NODE

Graph node i's feature $\boldsymbol{v}_i \in \mathbb{R}^{2d+1}$ can be thought of as a concatenation of three components $[\underbrace{\boldsymbol{x}_i}_{d \text{ dim}}, \underbrace{\mathsf{tmp}}_{d \text{ dim}}, \underbrace{\mathsf{partialsum}}_{1 \text{ dim}}]$, where $\boldsymbol{x}_i, \in \mathbb{R}^d$, $\mathsf{tmp} \in \mathbb{R}^d$ [3], and $\mathsf{partialsum} \in \mathbb{R}$.

In particular, $\boldsymbol{x}_i$ is the initial node feature. The first d channel will stay the same until the layer $n+2$. $\mathsf{tmp} = \sum_{j \in \text{subset of}[n]} e^{\alpha'_{ij}} \boldsymbol{x}_j$ stands for the unnormalized attention contribution up to the current iteration. $\mathsf{partialsum} \in \mathbb{R}$ is a partial sum of the unnormalized attention score, which will be used for normalization in the $n+2$-th iteration.

Initial feature $\boldsymbol{z}_{\text{gn}}^{(0)} = [\boldsymbol{x}_i, \mathbf{0}_d, 0]$.

Message function + Aggregate function: $\tau_{j \in [n]} \phi_{gn\text{-}vn}^{(k)} : \mathbb{R}^{2d+1} \times \mathbb{R}^{2d+1} \to \mathbb{R}^{2d+1}$ is just "copying the second argument" since there is just one incoming message from the virtual node, i.e., $\tau_{j \in [n]} \phi_{\text{gn-vn}}^{(k)}(\boldsymbol{x}, \{\boldsymbol{y}\}) = \boldsymbol{y}$. This function can be exactly implemented by an MLP.

Update function $\gamma_{\text{gn}}^{(k)} : \underbrace{\mathbb{R}^{2d+1}}_{\text{gn}} \times \underbrace{\mathbb{R}^{2d+1}}_{\text{vn}} \to \mathbb{R}^{2d+1}$ is of the following form.

$$\gamma_{\text{gn}}^{(k)}([\boldsymbol{x}, \mathsf{tmp}, \mathsf{partialsum}], \boldsymbol{y}) = \begin{cases} [\boldsymbol{x}, \mathsf{tmp}, \mathsf{partialsum}] & k = 1 \\ [\boldsymbol{x}, \mathsf{tmp} + e^{\alpha'(\boldsymbol{x}, \boldsymbol{y}_{0:d})} \boldsymbol{W}_V \boldsymbol{y}_{0:d}, & \\ \mathsf{partialsum} + e^{\alpha'(\boldsymbol{x}, \boldsymbol{y}_{0:d})}] & k = 2, ..., n+1 \\ [\frac{\mathsf{tmp}}{\mathsf{partialsum}}, \mathbf{0}_d, 0] & k = n+2 \end{cases} \tag{9}$$

where $\alpha'(\boldsymbol{x}, \boldsymbol{y}_{0:d})$ is the usual unnormalized attention score. Update function $\gamma_{\text{gn}}^{(k)}$ can be arbitrarily approximated by an MLP, which is proved below.

Lemma B.3. *Update function* $\gamma_{gn}^{(k)}$ *can be arbitrarily approximated by an MLP from* $\mathbb{R}^{2d+1} \times \mathbb{R}^{2d+1}$ *to* $\mathbb{R}^{2d+1}$ *for all* $k = 1, ..., n+2$.

Proof. We will show that for any $k = 1, ..., n+2$, the target function $\gamma_{\text{gn}}^{(k)} : \mathbb{R}^{2d+1} \times \mathbb{R}^{2d+1} \to \mathbb{R}^{2d+1}$ is continuous and the domain is compact. By the universality of MLP in approximating continuous function on the compact domain, we know $\gamma_{\text{gn}}^{(k)}$ can be approximated to arbitrary precision by an MLP.

Recall that

$$\gamma_{\text{gn}}^{(k)}([\boldsymbol{x}, \mathsf{tmp}, \mathsf{partialsum}], \boldsymbol{y}) = \begin{cases} [\boldsymbol{x}, \mathsf{tmp}, \mathsf{partialsum}] & k = 1 \\ [\boldsymbol{x}, \mathsf{tmp} + e^{\alpha'(\boldsymbol{x}, \boldsymbol{y}_{0:d})} \boldsymbol{W}_V \boldsymbol{y}_{0:d}, & \\ \mathsf{partialsum} + e^{\alpha'(\boldsymbol{x}, \boldsymbol{y}_{0:d})}] & k = 2, ..., n+1 \\ [\frac{\mathsf{tmp}}{\mathsf{partialsum}}, \mathbf{0}_d, 0] & k = n+2 \end{cases}$$

[3] tmp technicially denotes the dimension of projected feature by W_V and does not has to be in $\mathbb{R}^d$. We use $\mathbb{R}^d$ here to reduce the notation clutter.

it is easy to see that $k=1$, $\gamma_{\text{gn}}^{(1)}$ is continuous. We next show for $k=2,...,n+2$, $\gamma_{\text{gn}}^{(1)}$ is also continuous and all arguments lie in a compact domain.

$\gamma_{\text{gn}}^{(k)}$ is continuous because to a) $\alpha'(\boldsymbol{x},\boldsymbol{y})$ is continuous b) scalar-vector multiplication, sum, and exponential are all continuous. Next, we show that four component $\boldsymbol{x}$, tmp, $\mathsf{partialsum}$, $\boldsymbol{y}_{0:d}$ all lies in a compact domain.

$\boldsymbol{x}$ is the initial node features, and by AS1 their norm is bounded so $\boldsymbol{x}$ is in a compact domain.

tmp is an approximation of $e^{\alpha'_{i,1}}\boldsymbol{W}_V\boldsymbol{x}_1+e^{\alpha'_{i,2}}\boldsymbol{W}_V\boldsymbol{x}_2+....$ As $\alpha'(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is both upper and lower bounded by AS2 for all $i,j\in[n]$ and $\boldsymbol{x}_i$ is bounded by AS1, $e^{\alpha'_{i,1}}\boldsymbol{W}_V\boldsymbol{x}_1+e^{\alpha'_{i,2}}\boldsymbol{W}_V\boldsymbol{x}_2+...$ is also bounded from below and above. tmp will also be bounded as we can control the error to any precision.

$\mathsf{partialsum}$ is an approximation of $e^{\alpha'_{i,1}}+e^{\alpha'_{i,2}}+....$ For the same reason as the case above, $\mathsf{partialsum}$ is also bounded both below and above.

$\boldsymbol{y}_{0:d}$ will be $\tilde{\boldsymbol{x}}_i$ at i-th iteration so it will also be bounded by AS1.

Therefore we conclude the proof. □

B.3. A Running Example

We provide an example to illustrate how node features are updated in each iteration.

Time 0: All nodes are initialized as indicated in Appendix B.2. Virtual node feature $\boldsymbol{z}_{\text{vn}}^{(0)}=[\boldsymbol{0}_d,\boldsymbol{v}_1,0]$. Graph node feature $\boldsymbol{z}_i^{(0)}=[\boldsymbol{x}_i,\boldsymbol{0}_d,0]$ for all $i\in[n]$.

Time 1:

For virtual node, according to the definition of $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(1)}$ in Equation (7), it will pick an approximation of $\boldsymbol{x}_1$, i.e. $\tilde{\boldsymbol{x}}_1$. Note that the approximation error can be made arbitrarily small. VN's node feature $\boldsymbol{z}_{\text{vn}}^{(1)}=[\tilde{\boldsymbol{x}}_1,\boldsymbol{v}_2,0]$.

For i-th graph node feature, $\boldsymbol{z}_{\text{vn}}^{(0)}=\boldsymbol{1}_d$, and $\boldsymbol{z}_i^{(0)}=[\boldsymbol{x}_i,\boldsymbol{0}_d,0]$. According to $\gamma_{\text{gn}}^{(k)}$ in Equation (9), $\boldsymbol{z}_i^{(1)}=[\boldsymbol{x}_i,\boldsymbol{0}_d,0]$.

Time 2:

For the virtual node feature: similar to the analysis in time 1, VN's feature $\boldsymbol{z}_{\text{vn}}^{(2)}=[\tilde{\boldsymbol{x}}_2,\boldsymbol{v}_3,0]$ now. Note that the weights and bias in $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(2)}$ will be different from those in $\tau_{j\in[n]}\phi_{\text{vn-gn}}^{(1)}$.

For i-th graph node feature, as $\boldsymbol{z}_{\text{vn}}^{(1)}=[\tilde{\boldsymbol{x}}_1,\boldsymbol{v}_2,0]$ and $\boldsymbol{z}_i^{(1)}=[\boldsymbol{x}_i,\boldsymbol{0}_d,0]$, according to $\gamma_{\text{gn}}^{(k)}$ in Equation (9), $\boldsymbol{z}_i^{(2)}=[\boldsymbol{x}_i,e^{\widetilde{\alpha'_{i,1}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_1,e^{\widetilde{\alpha'_{i,1}}}]$. Here $\widetilde{\alpha'_{i,1}}:=\alpha'(\boldsymbol{x}_i,\tilde{\boldsymbol{x}}_1)$. We will use similar notations in later iterations. [4]

Time 3:

Similar to the analysis above, $\boldsymbol{z}_{\text{vn}}^{(3)}=[\widetilde{\boldsymbol{x}_3},\boldsymbol{v}_4,0]$.

$\boldsymbol{z}_i^{(3)}=[\boldsymbol{x}_i,e^{\widetilde{\alpha'_{i,1}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_1+e^{\widetilde{\alpha'_{i,2}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_2,e^{\widetilde{\alpha'_{i,1}}}+e^{\widetilde{\alpha'_{i,2}}}]$.

Time n:

$\boldsymbol{z}_{\text{vn}}^{(n)}=[\tilde{\boldsymbol{x}}_n,\boldsymbol{0}_d,0]$.

$\boldsymbol{z}_i^{(n)}=\boldsymbol{x}_i,\underbrace{e^{\widetilde{\alpha'_{i,1}}}\boldsymbol{W}_V\tilde{\boldsymbol{x}}_1+...+e^{\widetilde{\alpha'_{i,n-1}}}\boldsymbol{W}_V\widetilde{\boldsymbol{x}_{n-1}}}_{n-1\text{ terms}},$

$\underbrace{e^{\widetilde{\alpha'_{i,1}}}+e^{\widetilde{\alpha'_{i,2}}}+...+e^{\widetilde{\alpha'_{i,n-1}}}}_{n-1\text{ terms}}]$.

Time $n+1$:

[4] To reduce the notation clutter and provide an intuition of the proof, we omit the approximation error introduced by using MLP to approximate aggregation/message/update function, and assume the aggregation/message/update can be exactly implemented by neural networks. In the proofs, approximation error by MLP is handled rigorously.

According to Appendix B.2.1, in $n+1$ iteration, the virtual node's feature will be $\mathbf{1}_d$.

$\boldsymbol{z}_i^{(n+1)} = [\boldsymbol{x}_i, \sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}} \boldsymbol{W}_V \tilde{\boldsymbol{x}}_k, \sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}}]$

Time $n+2$ (final layer):

For the virtual node, its node feature will stay the same.

For the graph node feature, the last layer will serve as a normalization of the attention score (use MLP to approximate vector-scalar multiplication), and set the last channel to be 0 (projection), resulting in an approximation of $[\boldsymbol{x}_i, \frac{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}} \boldsymbol{W}_V \tilde{\boldsymbol{x}}_k}{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}}}, 0]$.

Finally, we need one more linear transformation to make the node feature become $[\frac{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}} \boldsymbol{W}_V \tilde{\boldsymbol{x}}_k}{\sum_{k\in[n]} e^{\widetilde{\alpha'_{ik}}}}, \mathbf{0}_d, 0]$. The first d channel is an approximation of the output of the self-attention layer for node i where the approximation error can be made as small as possible. This is proved in Appendix B, and we conclude that heterogeneous MPNN + VN can approximate the self-attention layer $\boldsymbol{L}$ to arbitrary precision with $\mathcal{O}(n)$ MPNN layers.

B.4. Controlling Error

On the high level, there are three major sources of approximation error: 1) approximate hard selection with self-attention and 2) approximate equation $\gamma_{\text{gn}}^{(k)}$ with MLPs, and 3) attention normalization in the last layer. In all cases, we aim to approximate the output of a continuous map $\boldsymbol{L}_c(\boldsymbol{x})$. However, our input is usually not exact $\boldsymbol{x}$ but an approximation of $\tilde{\boldsymbol{x}}$. We also cannot access the original map $\boldsymbol{L}_c$ but instead, an MLP approximation of $\boldsymbol{L}_c$, denoted as $\boldsymbol{L}_{\text{MLP}}$. The following lemma allows to control the difference between $\boldsymbol{L}_c(\boldsymbol{x})$ and $\boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})$.

Lemma B.4. *Let $\boldsymbol{L}_c$ be a continuous map from compact set to compact set in Euclidean space. Let $\boldsymbol{L}_{MLP}$ be the approximation of $\boldsymbol{L}_c$ by MLP. If we can control $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|$ to an arbitrarily small degree, we can then control the error $\|\boldsymbol{L}_c(\boldsymbol{x}) - \boldsymbol{L}_{MLP}(\tilde{\boldsymbol{x}})\|$ arbitrarily small.*

Proof. By triangle inequality $\|\boldsymbol{L}_c(\boldsymbol{x}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\| \leq \|\boldsymbol{L}_c(\boldsymbol{x}) - \boldsymbol{L}_{\text{MLP}}(\boldsymbol{x}))\| + \|\boldsymbol{L}_{\text{MLP}}(\boldsymbol{x}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$.

For the first term $\|\boldsymbol{L}_c(\tilde{\boldsymbol{x}}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$, by the universality of MLP, we can control the error $\|\boldsymbol{L}_c(\tilde{\boldsymbol{x}}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$ in arbitrary degree.

For the second term $\|\boldsymbol{L}_{\text{MLP}}(\boldsymbol{x}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$, as $\boldsymbol{L}_{\text{MLP}}$ is continuous on a compact domain, it is uniformly continuous by Heine-Cantor theorem. This means that we can control the $\|\boldsymbol{L}_{\text{MLP}}(\boldsymbol{x}) - \boldsymbol{L}_{\text{MLP}}(\tilde{\boldsymbol{x}})\|$ as long as we can control $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|$, independent from different $\boldsymbol{x}$. By assumption, this is indeed the case so we conclude the proof. □

Remark B.5. The implication is that when we are trying to approximate the output of a continuous map $\boldsymbol{L}_c$ on the compact domain by an MLP $\boldsymbol{L}_{\text{MLP}}$, it suffices to show the input is 1) $\|\boldsymbol{L}_c - \boldsymbol{L}_{\text{MLP}}\|_\infty$ and 2) $\|\tilde{\boldsymbol{x}} - \boldsymbol{x}\|$ can be made arbitrarily small. The first point is usually done by the universality of MLP on the compact domain (Cybenko, 1989). The second point needs to be shown case by case.

In the Appendix B.3, to simplify the notations we omit the error introduced by using MLP to approximate aggregation/message/update functions (continuous functions on the compact domain of $\mathbb{R}^d$.) in MPNN + VN. Lemma B.4 justify such reasoning.

Lemma B.6 ($\tilde{\boldsymbol{x}}_i$ approximates $\boldsymbol{x}_i$. $\widetilde{\alpha'_{i,j}}$ approximates $\alpha'_{i,j}$.)**.** *For any $\epsilon > 0$ and $x \in \mathcal{X}$, there exist a set of weights for message/aggregate functions of the virtual node such that $||\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i|| < \epsilon$ and $|\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}| < \epsilon$.*

Proof. By Lemma 6.2 We know that $\widetilde{\alpha_{i,j}} := \widetilde{\alpha}(\boldsymbol{x}_i, \boldsymbol{x}_j) \to \delta(i-j)$ as $C_3(\epsilon)$ goes to infinity. Therefore we have

$$||\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i|| = ||\sum_j \widetilde{\alpha_{i,j}} \boldsymbol{x}_j - \boldsymbol{x}_i|| = ||\sum (\widetilde{\alpha}_{i,j} - \delta(i-j))\boldsymbol{x}_j|| < \epsilon \sum ||\boldsymbol{x}_j|| < nC_1\epsilon \tag{10}$$

As n and C_1 are fixed, we can make the upper bound as small as we want by increasing C_3.

$|\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}| = |\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j) - \alpha'_{\text{MLP}}(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| = |\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j) - \alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| + |\alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j) - \alpha'_{\text{MLP}}(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| = |\alpha'(\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)| = (\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i)^T \boldsymbol{x}_j C_2^2 + \epsilon < nC_1\epsilon C_1 C_2^2 + \epsilon = (nC_1^2 C_2^2 + 1)\epsilon$. As $\alpha'_{i,j}, \widetilde{\alpha'_{i,j}}$ is bounded from above and below, it's easy to see

that $|e^{\alpha'_{i,j}} - e^{\widetilde{\alpha'_{i,j}}}| = |e^{\alpha'_{i,j}}(1 - e^{\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}})| < C(1 - e^{\alpha'_{i,j} - \widetilde{\alpha'_{i,j}}})$ can be controlled to arbitrarily degree. □

Theorem 6.3. *Assume AS 1-3 hold for the compact set $\mathcal{X}$ and $\boldsymbol{L}$. Given any graph G of size n with node features $\boldsymbol{X} \in \mathcal{X}$, and a self-attention layer $\boldsymbol{L}$ on G (fix $\boldsymbol{W}_K, \boldsymbol{W}_Q, \boldsymbol{W}_V$ in α), there exists a $\mathcal{O}(n)$ layer of heterogeneous MPNN + VN with the specific aggregate/update/message function that can approximate $\boldsymbol{L}$ on $\mathcal{X}$ arbitrarily well.*

Proof. i-th MPNN + VN layer will select $\tilde{\boldsymbol{x}}_i$, an arbitrary approximation i-th node feature $\boldsymbol{x}_i$ via attention mechanism. This is detailed in the message/aggregation function of the virtual node in Appendix B.2.1. Assuming the regularity condition on feature space $\mathcal{X}$, detailed in AS3, the approximation error can be made as small as needed, as shown in Lemmas 6.2 and B.6.

Virtual node will then pass the $\tilde{\boldsymbol{x}}_i$ to all graph nodes, which computes an approximation of $e^{\alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)}, \forall j \in [n]$. This step is detailed in the update function $\gamma^{(k)}_{\text{gn}}$ of graph nodes, which can also be approximated arbitrarily well by MLP, proved in Lemma B.3. By Lemma B.4, we have an arbitrary approximation of $e^{\alpha'(\tilde{\boldsymbol{x}}_i, \boldsymbol{x}_j)}, \forall j \in [n]$, which itself is an arbitrary approximation of $e^{\alpha'(\boldsymbol{x}_i, \boldsymbol{x}_j)}, \forall j \in [n]$.

Repeat such procedures n times for all graph nodes, we have an arbitrary approximation of $\sum_{k\in[n]} e^{\alpha'_{ik}} \boldsymbol{W}_V \boldsymbol{x}_k \in \mathbb{R}^d$ and $\sum_{k\in[n]} e^{\alpha'_{ik}} \in \mathbb{R}$. Finally, we use the last layer to approximate attention normalization $L_c(\boldsymbol{x}, y) = \frac{\boldsymbol{x}}{y}$, where $\boldsymbol{x} \in \mathbb{R}^d, y \in \mathbb{R}$. As inputs for attention normalization are arbitrary approximation of $\sum_{k\in[n]} e^{\alpha'_{ik}} \boldsymbol{W}_V \boldsymbol{x}_k$ and $\sum_{k\in[n]} e^{\alpha'_{ik}}$, both of them are lower/upper bounded according to AS1 and AS2. Since the denominator is upper bounded by a positive number, this implies that the target function L_c is continuous in both arguments. By evoking Lemma B.4 again, we conclude that we can approximate its output $\frac{\sum_{k\in[n]} e^{\alpha'_{ik}} \boldsymbol{W}_V \boldsymbol{x}_k}{\sum_{k\in[n]} e^{\alpha'_{ik}}}$ arbitrarily well. This concludes the proof.

□

B.5. Relaxing Assumptions with More Powerful Attention

One limitation of Theorem 6.3 are assumptions on node features space $\mathcal{X}$: we need to 1) restrict the variability of node feature so that we can select one node feature to process each iteration. 2) The space of the node feature also need to satisfy certain configuration in order for VN to select it. For 2), we now consider a different attention function for α_{vn} in MPNN + VN that can relax the assumptions AS3 on $\mathcal{X}$.

More powerful attention mechanism. From proof of Theorem 6.3, we just need $\alpha(\cdot, \cdot)$ uniformly select every node in $\boldsymbol{X} \in \mathcal{X}$. The unnormalized bilinear attention α' is weak in the sense that $f(\cdot) = \langle \boldsymbol{x}_i \boldsymbol{W}_Q \boldsymbol{W}_K^T, \cdot \rangle$ has a linear level set. Such a constraint can be relaxed via an improved attention module `GATv2`. Observing the ranking of the attention scores given by `GAT` (Veličković et al., 2017) is unconditioned on the query node, Brody et al. (2021) proposed `GATv2`, a more expressive attention mechanism. In particular, the unnormalized attention score $\alpha'_{\text{GATv2}}(\boldsymbol{u}, \boldsymbol{v}) := \boldsymbol{a}^T \,\text{LeakyReLU}\,(\boldsymbol{W} \cdot [\boldsymbol{u} \| \boldsymbol{v}] + \boldsymbol{b})$, where $[\cdot||\cdot]$ is concatenation. We will let $\alpha_{\text{vn}} = \alpha_{\text{GATv2}}$ to select features in $\tau_{j\in[n]} \phi^{(k)}_{\text{vn-gn}}$.

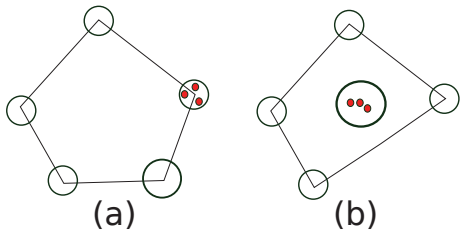

Figure 3: In the left figure, we have one example of $\mathcal{X}$ being $(\boldsymbol{V}, \delta)$ separable, for which α can uniformly select any point (marked as red) $\boldsymbol{x}_i \in \mathcal{X}_i$. In the right figure, we change α_{vn} in MPNN + VN to α_{GATv2}, which allows us to select more diverse feature configurations. The cluster in the middle cannot be selected by any $\alpha \in \mathcal{A}$ but can be selected by α_{GATv2} according to Proposition B.9.

Lemma B.7. *$\alpha'_{GATv2}(\cdot, \cdot)$ can approximate any continuous function from $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. For any $\boldsymbol{v} \in \mathbb{R}^d$, a restriction of $\alpha'_{GATv2}(\cdot, \boldsymbol{v})$ can approximate any continuous function from $\mathbb{R}^d \to \mathbb{R}$.*

Proof. Any function continuous in both arguments of α'_{GATv2} is also continuous in the concatenation of both arguments. As

any continuous functions in $\mathbb{R}^{2d}$ can be approximated by α'_{GATv2} on a compact domain according to the universality of MLP (Cybenko, 1989), we finish the proof for the first statement.

For the second statement, we can write W as 2×2 block matrix and restrict it to cases where only $\boldsymbol{W}_{11}$ is non-zero. Then we have

$$\alpha'_{\text{GATv2}}(\boldsymbol{u}, \boldsymbol{v}) = a^T \,\text{LeakyReLU}\left(\begin{bmatrix} \boldsymbol{W}_{11} & \boldsymbol{W}_{12} \\ \boldsymbol{W}_{21} & \boldsymbol{W}_{22} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} + \boldsymbol{b}\right) = \boldsymbol{a}^T \,\text{LeakyReLU}\left(\boldsymbol{W}_{11}\boldsymbol{u} + \boldsymbol{b}\right) \tag{11}$$

which gives us an MLP on the first argument $\boldsymbol{u}$. By the universality of MLP, we conclude the proof for the second statement.

□

Definition B.8. Given $\delta > 0$, We call $\mathcal{X}$ is δ nonlinearly separable if and only if $\min_{i \neq j} d(\mathcal{X}_i, \mathcal{X}_j) > \delta$.

AS 3'. *$\mathcal{X}$ is δ nonlinearly separable for some $\delta > 0$.*

Proposition B.9. *If $\mathcal{X} \subset \mathbb{R}^{n \times d}$ satisfies that $\mathcal{X}_i$ is δ-separated from $\mathcal{X}_j$ for any $i, j \in [n]$, the following holds. For any $\boldsymbol{X} \in \mathcal{X}$ and $i \in [n]$, there exist a α_{GATv2} to select any $\boldsymbol{x}_i \in \mathcal{X}_i$. This implies that we can arbitrarily approximate the self-attention layer $\boldsymbol{L}$ after relaxing AS3 to AS3'.*

Proof. For any $i \in [n]$, as $\mathcal{X}_i$ is δ-separated from other $\mathcal{X}_j, \forall j \neq i$, we can draw a region $\Omega_i \subset \mathbb{R}^d$ that contains $\mathcal{X}_i$ and separate $\mathcal{X}_i$ from other $\mathcal{X}_j (j \neq i)$, where the distance from $\mathcal{X}_i$ from other $\mathcal{X}_j$ is at least δ according to the definition of Definition B.8. Next, we show how to construct a continuous function f whose value in $\mathcal{X}_i$ is at least 1 larger than its values in any other $\mathcal{X}_j$ $\forall j \neq i$.

We set the values of f in $\mathcal{X}_i$ to be 1.5 and values of f in $\mathcal{X}_j, \forall j \neq i$ to be 0. We can then interpolate f in areas outside of $\cup \mathcal{X}_i$ (one way is to set the values of $f(x)$ based on $d(x, \mathcal{X}_i)$, which results in a continuous function that satisfies our requirement. By the universality of α_{GATv2}, we can approximate f to arbitrary precision, and this will let us select any $\mathcal{X}_i$. □

C. On the Limitation of MPNN + VN

Although we showed that in the main paper, MPNN + VN of varying depth/width can approximate the self-attention of full/linear transformers, this does not imply that there is no difference in practice between MPNN + VN and GT. Our theoretical analysis mainly focuses on approximating self-attention without considering computational efficiency. In this section, we mention a few limitations of MPNN + VN compared to GT.

C.1. Representation Gap

The main limitation of deep MPNN + VN approximating full self-attention is that we require a quite strong assumption: we restrict the variability of node features in order to select one node feature to process each iteration. Such assumption is relaxed by employing stronger attention in MPNN + VN but is still quite strong.

For the large width case, the main limitation is the computational complexity: even though the self-attention layer requires $\mathcal{O}(n^2)$ complexity, to approximate it in wide MPNN + VN framework, the complexity will become $\mathcal{O}(n^d)$ where d is the dimension of node features.

We think such limitation shares a similarity with research in universal permutational invariant functions. Both DeepSets (Zaheer et al., 2017) and Relational Network (Santoro et al., 2017) are universal permutational invariant architecture but there is still a representation gap between the two (Zweig & Bruna, 2022). Under the restriction to analytic activation functions, one can construct a symmetric function acting on sets of size n with elements in dimension d, which can be efficiently approximated by the Relational Network, but provably requires width exponential in n and d for the DeepSets. We believe a similar representation gap also exists between GT and MPNN + VN and leave the characterization of functions lying in such gap as the future work.

C.2. On The Difficulty of Approximating Other Linear Transformers

In Section 4, we showed MPNN + VN of $\mathcal{O}(1)$ width and depth can approximate the self-attention layer of one type of linear transformer, Performer. The literature on efficient transformers is vast (Tay et al., 2020) and we do not expect MPNN

+ VN can approximate many other efficient transformers. Here we sketch a few other linear transformers that are hard to approximate by MPNN + VN of constant depth and width.

Linformer (Wang et al., 2020b) projects the $n \times d$ dimension keys and values to $k \times d$ suing additional projection layers, which in graph setting is equivalent to graph coarsening. As MPNN + VN still operates on the original graph, it fundamentally lacks the key component to approximate Linformer.

We consider various types of efficient transformers effectively generalize the virtual node trick. By first switching to a more expansive model and reducing the computational complexity later on, efficient transformers effectively explore a larger model design space than MPNN + VN, which always sticks to the linear complexity.

C.3. Difficulty of Representing SAN Type Attention

In SAN (Kreuzer et al., 2021), different attentions are used conditional on whether an edge is presented in the graph or not, detailed below. One may wonder whether we can approximate such a framework in MPNN + VN.

In our proof of using MPNN + VN to approximate regular GT, we mainly work with Definition 3.4 where we do not use any gn-gn edges and therefore not leverage the graph topology. It is straightforward to use gn-gn edges and obtain the different message/update/aggregate functions for gn-gn edges non-gn-gn edges. Although we still achieve the similar goal of SAN to condition on the edge types, it turns out that we can not arbitrarily approximate SAN.

Without loss of generality, SAN uses two types of attention depending on whether two nodes are connected by the edge. Specifically,

$$\begin{aligned}
\hat{\boldsymbol{w}}_{ij}^{k,l} &= \left\{ \begin{array}{ll} \frac{\boldsymbol{Q}^{1,k,l}\boldsymbol{h}_i^l \circ \boldsymbol{K}^{1,k,l}\boldsymbol{h}_j^l \circ \boldsymbol{E}^{1,k,l}\boldsymbol{e}_{ij}}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\boldsymbol{Q}^{2,k,l}\boldsymbol{h}_i^l \circ \boldsymbol{K}^{2,k,l}\boldsymbol{h}_j^l \circ \boldsymbol{E}^{2,k,l}\boldsymbol{e}_{ij}}{\sqrt{d_k}} & \text{otherwise} \end{array} \right\} \\
w_{ij}^{k,l} &= \left\{ \begin{array}{ll} \frac{1}{1+\gamma} \cdot \mathrm{softmax}\left(\sum_{d_k} \hat{\boldsymbol{w}}_{ij}^{k,l}\right) & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\gamma}{1+\gamma} \cdot \mathrm{softmax}\left(\sum_{d_k} \hat{\boldsymbol{w}}_{ij}^{k,l}\right) & \text{otherwise} \end{array} \right\}
\end{aligned} \tag{12}$$

where $\circ$ denotes element-wise multiplication and $\boldsymbol{Q}^{1,k,l}, \boldsymbol{Q}^{2,k,l}, \boldsymbol{K}^{1,k,l}, \boldsymbol{K}^{2,k,l}, \boldsymbol{E}^{1,k,l}, \boldsymbol{E}^{2,k,l} \in \mathbb{R}^{d_k \times d}$. $\gamma \in \mathbb{R}^+$is a hyperparameter that tunes the amount of bias towards full-graph attention, allowing flexibility of the model to different datasets and tasks where the necessity to capture long-range dependencies may vary.

To reduce the notation clutter, we remove the layer index l, and edge features, and also consider only one-attention head case (remove attention index k). The equation is then simplified to

$$\begin{aligned}
\hat{\boldsymbol{w}}_{ij} &= \left\{ \begin{array}{ll} \frac{\boldsymbol{Q}^1\boldsymbol{h}_i^l \circ \boldsymbol{K}^1\boldsymbol{h}_j^l}{\sqrt{d_k}} & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\boldsymbol{Q}^2\boldsymbol{h}_i^l \circ \boldsymbol{K}^2\boldsymbol{h}_j^l}{\sqrt{d_k}} & \text{otherwise} \end{array} \right\} \\
w_{ij} &= \left\{ \begin{array}{ll} \frac{1}{1+\gamma} \cdot \mathrm{softmax}\left(\sum_d \hat{\boldsymbol{w}}_{ij}\right) & \text{if } i \text{ and } j \text{ are connected in sparse graph} \\ \frac{\gamma}{1+\gamma} \cdot \mathrm{softmax}\left(\sum_d \hat{\boldsymbol{w}}_{ij}\right) & \text{otherwise} \end{array} \right\}
\end{aligned} \tag{13}$$

We will then show that Equation (13) can not be expressed (up to an arbitrary approximation error) in MPNN + VN framework. To simulate SAN type attention, our MPNN + VN framework will have to first simulate one type of attention for all edges, as we did in the main paper, and then simulate the second type of attention between gn-gn edges by properly offset the contribution from the first attention. This seems impossible (although we do not have rigorous proof) as we cannot express the difference between two attention in the new attention mechanism.

D. Experimental Details

D.1. Dataset Description

ogbg-molhiv and **ogbg-molpcba** (Hu et al., 2020) are molecular property prediction datasets adopted by OGB from MoleculeNet. These datasets use a common node (atom) and edge (bond) featurization that represent chemophysical properties. The prediction task of ogbg-molhiv is a binary classification of molecule's fitness to inhibit HIV replication. The ogbg-molpcba, derived from PubChem BioAssay, targets to predict the results of 128 bioassays in the multi-task binary classification setting.

ogbg-ppa (Wu et al., 2021) consists of protein-protein association (PPA) networks derived from 1581 species categorized into 37 taxonomic groups. Nodes represent proteins and edges encode the normalized level of 7 different associations between two proteins. The task is to classify which of the 37 groups does a PPA network originate from.

ogbg-code2 (Wu et al., 2021) consists of abstract syntax trees (ASTs) derived from the source code of functions written in Python. The task is to predict the first 5 subtokens of the original function's name.

OGB-LSC PCQM4Mv2 (Hu et al., 2021) is a large-scale molecular dataset that shares the same featurization as ogbg-mol* datasets. It consists of 529,434 molecule graphs. The task is to predict the HOMO-LUMO gap, a quantum physical property originally calculated using Density Functional Theory. True labels for original test-dev and test-challange dataset splits are kept private by the OGB-LSC challenge organizers. Therefore for the purpose of this paper, we used the original validation set as the test set, while we left out random 150K molecules for our validation set.

D.2. Reproducibility

For LRGB results in Section 7.1, we reproduce the original results up to very small differences.

Table 7: Reproduce the original results up to small differences. No VN is used.

Model	# Params.	Peptides-func		Peptides-struct	
		Test AP (reproduce)	Test AP ↑	Test MAE (reproduce)	Test MAE ↓
GCN	508k	0.5918±0.0065	0.5930±0.0023	0.3468±0.0009	0.3496±0.0013
GINE	476k	0.5595±0.0126	0.5498±0.0079	0.3532±0.0024	0.3547±0.0045
GatedGCN	509k	0.5886±0.0027	0.5864±0.0077	0.3409±0.0011	0.3420±0.0013
GatedGCN+RWSE	506k	0.6083±0.0032	0.6069±0.0035	0.3377±0.0025	0.3357±0.0006

D.3. The Role of Graph Topology

In our experiments, we considered graph topology in experiments (i.e., message passing operates on both GN-VN (graph node-virtual node) and GN-GN edges). To understand the role of GN-VN and GN-GN edges, we carried out a set of new experiments where we discard the original graph topology, and only do message passing on GN-VN edges, for Peptides-func & Peptides-struct datasets. The results are shown in Appendix D.3.

We observe that in general, MPNN + VN using GN-VN edges only perform slightly worse than MPNN + VN using both GN-VN and GN-GN edges. However, it still performs better than the standard MPNN without VN. We believe adding VN as a simple way of long-range modeling is the main reason we see good results on Peptides-func & Peptides-struct datasets. Utilizing local graph topology in MPNN will further improve the performance.

In general, combining local (message passing) and global modeling (such as GT and VN) in GNN is an active research direction, with novel applications in macromolecule (DNA, RNA, Protein) modeling. In the recent SOTA model GraphGPS (Rampášek et al., 2022), MPNN is interleaved with GT. Consistent with our findings, Rampášek et al. (2022) also showed both the local component (MPNN) and global component (GT) contribute to the final performance.

Table 8: Utilizing local graph topology in MPNN will further improve the performance on `Peptides-func` and `Peptides-struct`.

	Peptides-func AP ↑			Peptides-struct MAE ↓		
	w/o VN (only graph topology)	w/ VN + graph topology	Only VN	w/o VN (only graph topology)	w/ VN + graph topology	Only VN
GCN	0.5930 ± 0.0023	0.6623 ± 0.0038	0.6488 ± 0.0056	0.3496 ± 0.0013	0.2488 ± 0.0021	0.2511 ± 0.0025
GINE	0.5498 ± 0.0079	0.6346 ± 0.0071	0.6022 ± 0.0072	0.3547 ± 0.0045	0.2584 ± 0.0011	0.2608 ± 0.0021
GatedGCN	0.5864 ± 0.0077	0.6635 ± 0.0024	0.6493 ± 0.0044	0.3420 ± 0.0013	0.2523 ± 0.0016	0.2684 ± 0.0039
GatedGCN+RWSE	0.6069 ± 0.0035	0.6685 ± 0.0062	0.6432 ± 0.0072	0.3357 ± 0.0006	0.2529 ± 0.0009	0.2645 ± 0.0023

D.4. Additional Experiments

We tested MPNN + VN on `PascalVOC-SP` datasets and also observe improvement, shown in Table 9, although the improvement is not as large as that of `Peptides-func` and `Peptides-struct` datasets. The best MPNN + VN model

is GatedGCN + LapPE where the performance gap to the best GT model is rather small.

Table 9: Baseline experiments for `PascalVOC-SP` and `COCO-SP` with `rag-boundary` graph on SLIC compactness 30 for the node classification task. The performance metric is macro F1 on the respective splits (Higher is better). All experiments are run 4 times with 4 different seeds. The MP-GNN models are 8 layers deep, while the transformer-based models have 4 layers in order to maintain comparable hidden representation size at the fixed parameter budget of 500k. **Bold**: Best score.

Model	# Params	PascalVOC-SP	
		Before VN + Test F1	**After VN + Test F1 ↑**
GCN	496k	0.1268±0.0060	0.1901±0.0040
GINE	505k	0.1265±0.0076	0.1198±0.0073
GatedGCN	502k	0.2873±0.0219	0.2874±0.0178
GatedGCN+LapPE	502k	0.2860±0.0085	**0.3103±0.0068**
Transformer+LapPE	501k	0.2694±0.0098	-
SAN+LapPE	531k	**0.3230±0.0039**	-
SAN+RWSE	468k	0.3216±0.0027	-

D.5. Predicting Sea Surface Temperature

In this experiment, we consider a specific physical modeling problem: forecasting sea surface temperature (SST), that is the water temperature close to the ocean's surface. SST is an essential climate indicator and plays a significant role in analyzing and monitoring the dynamics of weather, climate, and other biological systems for several applications in environmental protection, agriculture, and industry. We use the NOAA/NESDIS/NCEI Daily Optimum Interpolation Sea Surface Temperature (DOISST) version 2.1 proposed by (Huang et al., 2021) as an improvement upon version 2.0 from (Reynolds et al., 2007).

We consider the daily SST data of the Pacific Ocean from 1982 to 2021, in the region of longitudes from $180.125°$E to $269.875°$E and latitudes from $-14.875°$N to $14.875°$N. We reduce the resolution of the original data from $0.25°$-degree to $0.5°$-degree. Following the procedure from (de Bezenac et al., 2018), (de Bézenac et al., 2019) and (Wang et al., 2022), we divide the region into 11 square batches of equal size (see Table 11), each contains exactly 30 longitudes and 30 latitudes that can be represented as a grid graph of 900 nodes in which we connect each node to its nearest 8 neighbors. We take time series from 1982 to 2018 as our training set, data in 2019 as our validation set, and data from 2020 to 2021 as our testing set. In our experiments, we set the history window w_h as 6 weeks (i.e. 42 days) and the prediction window w_p as 4 weeks (i.e. 28 days), 2 weeks (i.e. 14 days) or 1 week (i.e. 7 days). For each example, each node of the graph is associated with an input time series capturing the temperatures at the corresponding (longitude, latitude) for the last w_h days, and the task is to predict the output time series of temperatures for the next w_p days.

We represent each time series as a long vector and the learning task is fundamentally a node-level regression task. We make sure that there is no overlapping among training, validation and testing sets (e.g., the output of a training example will *not* appear in any input of another validation example). The number of training, validation, and testing examples are roughly 150K, 3K and 7K, respectively for each setting (see Table 10). We compare our MPNN + VN model with:

- Multilayer Perceptron (MLP) which treats both the input and output as long vectors and has 512 hidden neurons.
- TF-Net (Wang et al., 2020a) with the setting as in the original paper.
- Linear Transformer (Katharopoulos et al., 2020a) (Wang et al., 2020b)[5] with Laplacian positional encoding (LapPE). We compute the first 16 eigenvectors as positions for LapPE.

Both MPNN and MPNN + VN have 3 layers of message passing with 256 hidden dimensions. We apply an MLP with one hidden layer of 512 neurons on top of the network to make the final prediction.

[5]The Linear Transformer implementation is publicly available at `https://github.com/lucidrains/linear-attention-transformer`

Table 10: Number of training, validation and testing examples for each setting in the task of SST prediction.

History window	Prediction window	Train size	Validation size	Test size
6 weeks	4 weeks	$147,884$	$3,245$	$7,271$
	2 weeks	$148,038$	$3,399$	$7,425$
	1 week	$148,115$	$3,476$	$7,502$

Table 11: These are 11 regions of the Pacific in our experiment.

Index	Longitudes	Latitues
1	[180.125°E, 194.875°E]	[-14.875°N, -0.125°N]
2	[195.125°E, 209.875°E]	[-14.875°N, -0.125°N]
3	[210.125°E, 224.875°E]	[-14.875°N, -0.125°N]
4	[225.125°E, 239.875°E]	[-14.875°N, -0.125°N]
5	[240.125°E, 254.875°E]	[-14.875°N, -0.125°N]
6	[255.125°E, 269.875°E]	[-14.875°N, -0.125°N]
7	[180.125°E, 194.875°E]	[0.125°N, 14.875°N]
8	[195.125°E, 209.875°E]	[0.125°N, 14.875°N]
9	[210.125°E, 224.875°E]	[0.125°N, 14.875°N]
10	[225.125°E, 239.875°E]	[0.125°N, 14.875°N]
11	[240.125°E, 254.875°E]	[0.125°N, 14.875°N]

We train all our models with 100 epochs with batch size 20, initial learning rate 10^{-3}, and Adam optimizer (Kingma & Ba, 2014).

D.6. Connection to Over-Smoothing Phenomenon

Over-smoothing refers to the phenomenon that deep GNN will produce same features at different nodes after too many convolution layers. Here we draw some connection between VN and common ways of reducing over-smoothing. We think that using VN can potentially help alleviate the over-smoothing problem. In particular, we note that the use of VN can simulate some strategies people use in practice to address over-smoothing. We give two examples below.

Example 1: In (Zhao & Akoglu, 2019), the two-step method (center & scale) PairNorm is proposed to reduce the over-smoothing issues. In particular, PairNorm consists of 1) Center and 2) Scale

$$\tilde{\mathbf{x}}_i^c = \tilde{\mathbf{x}}_i - \frac{1}{n}\sum_i \tilde{\mathbf{x}}_i$$

$$\dot{\mathbf{x}}_i = s \cdot \frac{\tilde{\mathbf{x}}_i^c}{\sqrt{\frac{1}{n}\sum_i ||\tilde{\mathbf{x}}_i^c||_2^2}}$$

Where $\tilde{\mathbf{x}}$ is the node features after graph convolution and s is a hyperparameter. The main component for implementing PairNorm is to compute the mean and standard deviation of node features. For the mean of node features, this can be exactly computed in VN. For standard deviation, VN can arbitrarily approximate it using the standard universality result of MLP [5]. If we further assume that the standard deviation is lower bounded by a constant, then MPNN + VN can arbitrarily approximate the PairNorm on the compact set.

Example 2: In (Yang et al., 2020) mean subtraction (same as the first step of PairNorm) is also introduced to reduce over-smoothing. As mean subtraction can be trivially implemented in MPNN + VN, arguments in (Yang et al., 2020) (with mean subtraction the revised power Iteration in GCN will lead to the Fiedler vector) can be carried over to MPNN + VN setting.

In summary, introducing VN allows MPNN to implement key components of (Yang et al., 2020; Zhao & Akoglu, 2019), we think this is one reason why we observe encouraging empirical performance gain of MPNN + VN.