

# A GENERALIZATION OF ViT/MLP-MIXER TO GRAPHS

Xiaoxin He<sup>1</sup> Bryan Hooi<sup>1</sup> Thomas Laurent<sup>2</sup> Adam Perold<sup>3</sup> Yann LeCun<sup>4,5</sup> Xavier Bresson<sup>1</sup>

{xiaoxin, bhooi, xaviercs}@comp.nus.edu.sg, tlaurent@lmu.edu  
ap@discoverelement.com, yann@cs.nyu.edu

<sup>1</sup>National University of Singapore <sup>2</sup>Loyola Marymount University <sup>3</sup>Element, Inc.

<sup>4</sup>New York University <sup>5</sup>Meta AI

## ABSTRACT

Graph Neural Networks (GNNs) have shown great potential in the field of graph representation learning. Standard GNNs define a local message-passing mechanism which propagates information over the whole graph domain by stacking multiple layers. This paradigm suffers from two major limitations, over-squashing and poor long-range dependencies, that can be solved using global attention but significantly increases the computational cost to quadratic complexity. In this work, we propose an alternative approach to overcome these structural limitations by leveraging the ViT/MLP-Mixer architectures introduced in computer vision. We introduce a new class of GNNs, called Graph MLP-Mixer, that holds three key properties. First, they capture long-range dependency and mitigate the issue of over-squashing as demonstrated on the Long Range Graph Benchmark (LRGB) and the TreeNeighbourMatch datasets. Second, they offer better speed and memory efficiency with a complexity linear to the number of nodes and edges, surpassing the related Graph Transformer and expressive GNN models. Third, they show high expressivity in terms of graph isomorphism as they can distinguish at least 3-WL non-isomorphic graphs. We test our architecture on 4 simulated datasets and 7 real-world benchmarks, and show highly competitive results on all of them.<sup>1</sup>

## 1 Message-Passing GNNs and their Limitations

In this first section, we present the background of the project by introducing the standard Message-Passing (MP) GNNs and their two major limitations; low expressivity representation and poor long-range dependency. We also present the current techniques that address these issues, i.e. Weisfeiler-Leman GNNs, graph positional encoding and Graph Transformers, as well as their shortcomings.

**Message-Passing GNNs (MP-GNNs).** GNNs have become the standard learning architectures for graphs based on their flexibility to work with complex data domains s.a. recommendation [1, 2], chemistry [3, 4], physics [5, 6], transportation [7], vision [8], NLP [9], knowledge graphs [10], drug design [11, 12] and medical domain [13, 14]. Most GNNs are designed to have two core components. First, a structural message-passing mechanism s.a. [15, 16, 17, 1, 18, 4, 19] that computes node representations by aggregating the local 1-hop neighborhood information. Second, a stack of  $L$  layers that aggregates  $L$ -hop neighborhood nodes to increase the expressivity of the network and transmit information between nodes that are  $L$  hops apart.

**Weisfeiler-Leman GNNs (WL-GNNs).** One of the major limitations of MP-GNNs is their inability to distinguish (simple) non-isomorphic graphs. This limited expressivity can be formally analyzed with the Weisfeiler-Leman graph isomorphism test [20], as first proposed in Xu et al. [21], Morris et al. [22]. Later on, Maron et al. [23] introduced a general class of  $k$ -order WL-GNNs that can be proved to universally represent any class of  $k$ -WL graphs [24, 25]. But to achieve such expressivity, this class of GNNs requires using  $k$ -tuples of nodes with memory and speed complexities of  $O(N^k)$ , with  $N$  being the number of nodes and  $k \geq 3$ . Although the complexity can be reduced to  $O(N^2)$  and  $O(N^3)$  respectively [24, 25, 26], it is still computationally costly compared to the linear complexity  $O(E)$  of MP-GNNs with  $E$  being the number of edges, which often reduces to  $O(N)$  for real-world graphs that exhibit sparse structures s.a. molecules, knowledge graphs, transportation networks, gene regulatory networks, to name a few. In order to reduce memory and speed complexities of WL-GNNs while keeping high expressivity, several works have focused on

<sup>1</sup>Code: <https://github.com/XiaoxinHe/Graph-MLPMixer>

designing graph networks from their sub-structures s.a. sub-graph isomorphism [27], sub-graph routing mechanism [28], cellular WL sub-graphs [29], and k-hop egonet sub-graphs [21, 30, 25, 31, 32].

**Graph Positional Encoding (PE).** Another aspect of the limited expressivity of GNNs is their inability to recognize simple graph structures s.a. cycles or cliques, which are often present in molecules and social graphs [33]. We can consider  $k$ -order WL-GNNs with value  $k$  to be the length of cycle/clique, but with high complexity  $O(N^k)$ . An alternative approach is to add positional encoding to the graph nodes. It was proved in Murphy et al. [34], Loukas [35] that unique and equivariant PE increases the representation power of any MP-GNN while keeping the linear complexity. This theoretical result was applied with great empirical success with index PE [34], Laplacian eigenvectors [36, 37, 38, 39] and k-step Random Walk [40, 41]. All these graph PEs lead to GNNs strictly more powerful than the 1-WL test, which seems to be enough expressivity in practice [42]. However, none of the PE proposed for graphs can provide a global position of the nodes that is unique, equivariant and distance sensitive. This is due to the fact that a canonical positioning of nodes does not exist for arbitrary graphs, as there is no notion of up, down, left and right on graphs. For example, any embedding coordinate system like graph Laplacian eigenvectors [43] can flip up-down directions, right-left directions, and would still be a valid PE. This introduces ambiguities for the GNNs that require to (learn to) be invariant with respect to the graph or PE symmetries. A well-known example is given by the eigenvectors: there exist  $2^k$  number of possible sign flips for  $k$  eigenvectors that require to be learned by the network.

**Issue of long-range dependencies.** Another major limitation of MP-GNNs is the well-known issue of long-range dependencies. Standard MP-GNNs require  $L$  layers to propagate the information from one node to their  $L$ -hop neighborhood. This implies that the receptive field size for GNNs can grow exponentially, for example with  $O(2^L)$  for binary tree graphs. This causes over-squashing; exponentially growing information is compressed into a fixed-length vector by the aggregation mechanism [44, 45]. It is worth noting that the poor long-range modeling ability of deep GNNs can be caused by the combined effect of multiple factors, such as over-squashing, vanishing gradients, poor isomorphism expressivity, etc. but, in this work, we focus our effort on alleviating over-squashing s.a. [46, 47]. Over-squashing is well-known since recurrent neural networks [48], which have led to the development of the (self- and cross-)attention mechanisms for the translation task [49, 50] first, and then for more general natural language processing (NLP) tasks [51, 52]. Transformer architectures are the most elaborated networks that leverage attention, and have gained great success in NLP and computer vision (CV). Several works have generalized the transformer architecture for graphs, alleviating the issue of long-range dependencies and achieving competitive or superior performance against standard MP-GNNs. We highlight the most recent research works in the next paragraph.

**Graph Transformers.** GraphTransformers [37] generalize Transformers to graphs, with graph Laplacian eigenvectors as node PE, and incorporating graph structure into the permutation-invariant attention function. SAN and LSPE [38, 41] further improve with PE learned from Laplacian and random walk operators. GraphiT [53] encodes relative PE derived from diffusion kernels into the attention mechanism. GraphTrans [54] and SAT [55] add Transformers on the top of standard GNN layers. Graphormer [56] introduce three structural encodings, with great success on large molecular benchmarks. GPS [57] categorizes the different types of PE and puts forward a hybrid MPNN+Transformer architecture. We refer to Min et al. [58] for an overview of graph-structured Transformers. Generally, most Graph Transformer architectures address the problems of over-squashing and limited long-range dependencies in GNNs but they also increase significantly the complexity from  $O(E)$  to  $O(N^2)$ , resulting in a computational bottleneck.

## 2 Generalizing ViT/ML-Mixer to Graphs Overcome MP-GNN Limitations

In the following, we explain the importance of generalizing the ViT/MLP-Mixer architectures from computer vision to graphs. Precisely, this generalization is required to achieve three key properties for GNNs simultaneously which are 1) capturing long-range interaction, 2) keeping low linear speed/memory complexity as MP-GNNs and 3) offering high expressivity to distinguish non-isomorphic graphs. To the best of our knowledge, the proposed work is one of the first GNN architectures that can provide a substantial trade-off between capturing significant long-range dependency and low computational complexity, hence going beyond standard MP-GNNs.

**ViT and MLP-Mixer in Computer Vision.** Transformers have gained remarkable success in CV and NLP, most notably with architectures like ViT [59] and BERT [51]. The success of transformers has been long attributed to the attention mechanism [50], which is able to model long-range dependency by making "everything connected to everything". But recently, this prominent line of networks has been challenged by more cost efficient alternatives. A novel family of models based on the MLP-Mixer introduced by Tolstikhin et al. [60] has emerged and gained recognition for its simplicity and its efficient implementation. Overall, MLP-Mixer replaces the attention module with multi-layer perceptrons (MLPs) which are also not affected by over-squashing and poor long-range interaction. The original architecture is simple [60]; it takes image patches (or tokens) as inputs, encodes them with a linear layer (equivalent to a convolutional layer over the image patches), and updates their representations with a series of

feed-forward layers applied alternatively to image patches (or tokens) and features. The follow-up variants investigate different mixing operations, such as ResMLP [61], gMLP [62], and DynaMixer [63]. These plain networks can perform competitively with state-of-the-art (SOTA) vision Transformers, which tends to indicate that attention is not the only important inductive bias, but other elements like the general architecture of Transformers with patch embedding, residual connection and layer normalization, and carefully-curated data augmentation techniques seem to play essential roles as well [64].

**The Benefits of Generalizing MLP-Mixer from Grids to Graphs.** The original MLP-Mixer architecture is designed to capture long-range interaction between image components while keeping a low-computational cost. In this work, we will transfer these architectural advantages to GNNs, which requires generalizing the MLP-Mixer initially developed for grids and sequences to arbitrary graph topology. However, achieving such a successful generalization is challenging given the irregular and variable nature of graphs. See Section 3 for a detailed presentation of these challenges.

**Contribution.** The main contribution of this project is to design a novel GNN architecture that simultaneously captures long-range interaction, keeps low computational complexity, and is isomorphically expressive. Standard MP-GNNs have linear learning/inference complexities but low representation power and poor long-range dependency. Graph Transformers address these two problems but lose the computational efficiency with a quadratic complexity price. A generalization of MLP-Mixer to graphs overcomes the computational bottleneck of Graph Transformers and solves the issue of long-distance dependency. Our contributions are listed as follows.

- We identify the key challenges to generalize MLP-Mixer from images to graphs and design a new efficient class of GNNs, namely Graph MLP-Mixer, that simultaneously captures long-range dependency, keeps linear speed/memory complexity, and achieves high graph isomorphic expressivity.
- We show competitive results on multiple benchmarks from Benchmarking GNNs [36] and the Open Graph Benchmark (OGB) [65]; specifically, with 0.075 MAE on ZINC and 0.8073 ROCAUC on MolHIV.
- We demonstrate the capacity of the proposed model to capture long-range dependency with SOTA performance on two LRGB datasets [66] while keeping low complexity, to mitigate the over-squashing issue on the TreeNeighbourMatch dataset [44], and to reach the 3-WL expressive power on the SR25 dataset [67].
- Our approach forms a bridge between CV, NLP and graphs under a unified architecture, that can potentially benefit cross-over domain collaborations to design better networks.

### 3 Generalization Challenges

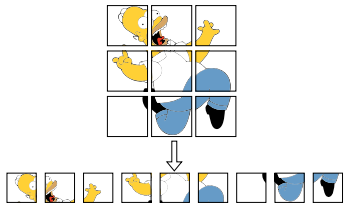
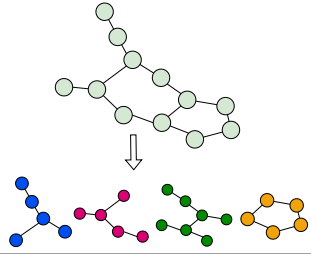
In the following, we list the main questions when adapting MLP-Mixer from images to graphs and summarize the differences between standard MLP-Mixer and Graph MLP-Mixer in Table 1.

**(1) How to define and extract graph patches/tokens?** One notable geometrical property that distinguishes graph-structured data from regular structured data, such as images and sequences, is that there does not exist in general a canonical grid to embed graphs. As shown in Table 1, images are supported by a regular lattice, which can be easily split into multiple grid-like patches (also referred to as tokens) of the same size via fast pixel reordering. However, graph data is irregular: the number of nodes and edges in different graphs is typically different. Hence, graphs cannot be uniformly divided into similar patches across all examples in the dataset. Finally, the extraction process for graph patches cannot be uniquely defined given the lack of canonical graph embedding. This raises the questions of how we identify meaningful graph tokens, and quickly extract them.

**(2) How to encode graph patches into a vectorial representation?** Since images can be reshaped into patches of the same size, they can be linearly encoded with an MLP, or equivalently with a convolutional layer with kernel size and stride values equal to the patch size. However, graph patches are not all the same size: they have variable topological structure with different number of nodes, edges and connectivity. Another important difference is the absence of a unique node ordering for graphs, which constrains the process to be invariant to node re-indexing for generalization purposes. In summary, we need a process that can transform graph patches into a fixed-length vectorial representation for arbitrary subgraph structures while being permutation invariant. GNNs are naturally designed to perform such transformations, and as such will be used to encode graph patches.

**(3) How to preserve positional information for nodes and graph patches?** As shown in Table 1, image patches in the sequence have implicit positions since image data is always ordered the same way due to its unique embedding in the Euclidean space. For instance, the image patch at the upper-left corner is always the first one in the sequence and the image patch at the bottom-right corner is the last one. On this basis, the token mixing operation of the MLP-Mixer is able to fuse the same patch information. However, graphs are naturally not-aligned and the set of graph patches are therefore unordered. We face a similar issue when we consider the positions of nodes within each graph patch. In images, the pixels in each patch are always ordered the same way; in contrast, nodes in graph tokens are naturally

Table 1: Differences between MLP-Mixer components for images and graphs.

	Images	Graphs
		
Input	Regular grid Same data resolution (Height, Width)	Irregular domain Variable data structure (# Nodes and # Edges)
Patch Extraction	Via pixel reordering Non-overlapping patches Same patches at each epoch	Via graph clustering algorithm Overlapping patches Different patches at each epoch
Patch Encoder	Same patch resolution (Patch Height, Patch Width) MLP (equivalently CNN)	Variable patch structure (# Nodes and # Edges) GNN (e.g. GCN, GAT, GT)
Positional Information	Implicitly ordered (No need for explicit PE)	No universal ordering Node PE for patch encoder Patch PE for token mixer
MLP-Mixer	Channel mixer Token mixer	Channel mixer Token mixer with patch PE

unordered. Thus, how do we preserve local positional consistency for nodes in each patch and global positional consistency for graph patches?

**(4) How to reduce over-fitting for Graph MLP-Mixer?** MLP-Mixer architectures are known to be strong over-fitters [62]. Most MLP-variants [60, 61, 63] first pre-train on large-scale datasets, and then fine-tune on downstream tasks, coupled with a rich set of data augmentation and regularization techniques, e.g. cropping, random horizontal flipping, RandAugment [68], mixup [69], etc. While data augmentation has drawn much attention in CV and NLP, graph data augmentation methods are not yet as effective, albeit interest and works on this topic [70]. Variable number of nodes, edges and connectivity make graph augmentation challenging. Thus, how do we augment graph-structured data given this nature of graphs?

## 4 Proposed Architecture

### 4.1 Overview

The basic architecture of Graph MLP-Mixer is illustrated in Figure 1. The goal of this section is to detail the choices we made to implement each component of the architecture. On the whole, these choices lead to a simple framework that provides speed and quality results.

**Notation.** Let  $G = (\mathcal{V}, \mathcal{E})$  be a graph with  $\mathcal{V}$  being the set of nodes and  $\mathcal{E}$  the set of edges. The graph has  $N = |\mathcal{V}|$  nodes and  $E = |\mathcal{E}|$  edges. The connectivity of the graph is represented by the adjacency matrix  $A \in \mathbb{R}^{N \times N}$ . The node features of node  $i$  are denoted by  $h_i$ , while the features for an edge between nodes  $i$  and  $j$  are indicated by  $e_{ij}$ . Let  $\{\mathcal{V}_1, \dots, \mathcal{V}_P\}$  be the nodes partition,  $P$  be the pre-defined number of patches, and  $G_i = (\mathcal{V}_i, \mathcal{E}_i)$  be the induced subgraph of  $G$  with all the nodes in  $\mathcal{V}_i$  and all the edges whose endpoints belong to  $\mathcal{V}_i$ . Let  $h_G$  be the graph-level vectorial representation and  $y_G$  be the graph-level target, which can be a discrete variable for graph classification problem, or a scalar for graph regression task.

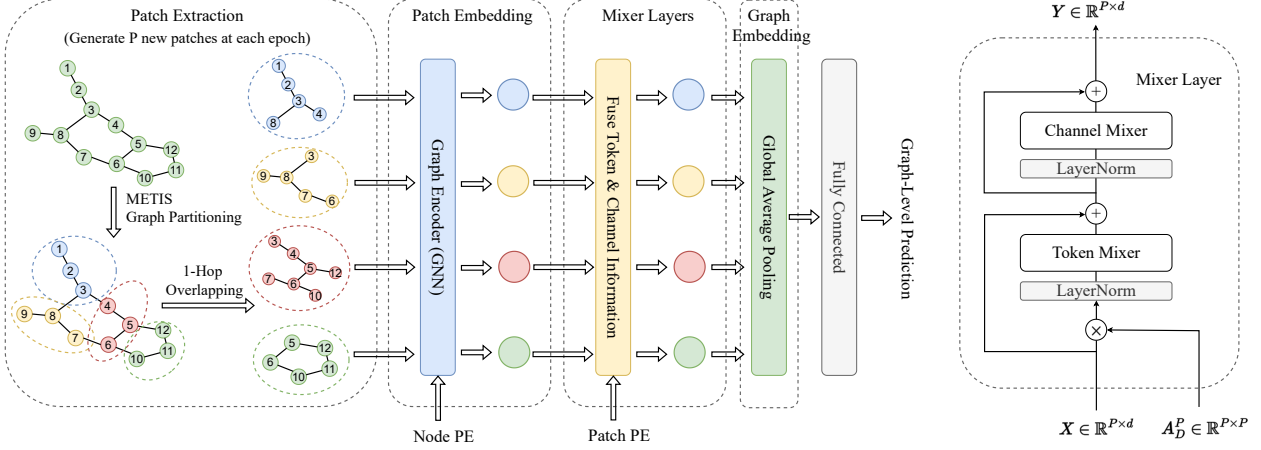


Figure 1: The basic architecture of the proposed Graph MLP-Mixer. Graph MLP-Mixer consists of a patch extraction module, a patch embedding module, a sequence of mixer layers, a global average pooling, and a classifier head. The patch extraction module partitions graphs into overlapping patches. The patch embedding module transforms these graph patches into corresponding token representations, which are fed into a sequence of mixer layers to generate the output tokens. A global average pooling layer followed by a fully-connected layer is finally used for prediction. Each Mixer Layer is a residual network that alternates between a Token Mixer applied to all patches, and a Channel Mixer applied to each patch independently (see right side).

## 4.2 Patch Extraction

When generalizing MLP-Mixer to graphs, the first step is to extract patches. This extraction is straightforward for images. Indeed, all image data  $x \in \mathbb{R}^{H \times W \times C}$  are defined on a regular grid with the same fixed resolution ( $H, W$ ), where  $H$  and  $W$  are respectively the height and the width, and  $C$  is the number of channels. Hence, all images can be easily reshaped into a sequence of flattened patches  $x_p \in \mathbb{R}^{P \times (R^2 C)}$ , where  $(R, R)$  is the resolution of each image patch, and  $P = HW/R^2$  is the resulting number of patches, see Table 1.

Unlike images with fixed resolution, extracting graph patches is more challenging. Generally, graphs have different sizes, i.e. number of nodes, and therefore cannot be uniformly divided like image data. Additionally, meaningful sub-graphs must be identified in the sense that nodes and edges composing a patch must share similar semantic or information, s.a. a community of friends sharing biking interest in a social network. As such, a graph patch extraction process must satisfy the following conditions: (1) the same extraction algorithm can be applied to any arbitrary graph, (2) the nodes in the sub-graph patch must be more closely connected than for those outside the patch, and (3) the extraction complexity must be fast, that is at most linear w.r.t. the number of edges, i.e.  $O(E)$ .

Graph partitioning algorithms have been studied for decades [71] given their importance in identifying meaningful clusters. Mathematically, graph partitioning is known to be NP-hard [72]. Approximations are thus required. A graph clustering algorithm with one of the best trade-off accuracy and speed is METIS [73], which partitions a graph into a pre-defined number of clusters/patches such that the number of within-cluster links is much higher than between-cluster links in order to better capture good community structure. For these fine properties, we select METIS as our graph patch extraction algorithm.

However, METIS is limited to finding non-overlapping clusters, as visualized in Figure 1. In this example, METIS partitions the graph into four non-overlapping parts, i.e.  $\{1, 2, 3\}$ ,  $\{4, 5, 6\}$ ,  $\{7, 8, 9\}$  and  $\{10, 11, 12\}$ , resulting in 5 edge cuts. Unlike images, extracting non-overlapping patches could imply losing important edge information, i.e. the cutting edges, and thus decreasing the predictive performance, as we will observe experimentally. To overcome this issue and to retain all original edges, we allow graph patches to overlap with each other. For example in Figure 1, if the source and destination nodes of an edge are not in the same patch, we assign both nodes to the patches they belong to. As such, node 3 and node 4 are in two different patches, here the blue and red one, but are connected with each other. After our overlapping adjustment, these two nodes belong to both the blue and red patches. This practice is equivalent to expanding the graph patches to the one-hop neighbourhood of all nodes in that patch. Formally, METIS is first applied to partition a graph into  $P$  non-overlapping patches:  $\{\mathcal{V}_1, \dots, \mathcal{V}_P\}$  such that  $\mathcal{V} = \mathcal{V}_1 \cup \dots \cup \mathcal{V}_P$  and  $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \forall i \neq j$ . Then, patches are expanded to their one-hop neighbourhood in order to preserve the information of between-patch links

and make use of all graph edges:  $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \{ \mathcal{N}_1(j) \mid j \in \mathcal{V}_i \}$ , where  $\mathcal{N}_k(j)$  defines the  $k$ -hop neighbourhood of node  $j$  and  $k = 1$  by default.

### 4.3 Patch Encoder

For images, patch encoding can be done with a simple linear transformation given the fixed resolution of all image patches. This operation is fast and well-defined. For graphs, the patch encoder network must be able to handle complex data structure such as invariance to index permutation, heterogeneous neighborhood, variable patch sizes, convolution on graphs, and expressive to differentiate graph isomorphisms. As a result, the graph patch encoder is a GNN, whose architecture is designed to best transform a graph token  $G_p$  into a fixed-size representation  $x_{G_p} \in \mathbb{R}^d$  into 3 steps.

**Step 1. Raw node and edge linear embedding.** The input node features  $\alpha_i \in \mathbb{R}^{d_n \times 1}$  and edge features  $\beta_{ij} \in \mathbb{R}^{d_e \times 1}$  are linearly projected into  $d$ -dimensional hidden features:

$$h_i^0 = U^0 \alpha_i + u^0 \in \mathbb{R}^d; \quad e_{ij}^0 = V^0 \beta_{ij} + v^0 \in \mathbb{R}^d \quad (1)$$

where  $U^0 \in \mathbb{R}^{d \times d_n}$ ,  $V^0 \in \mathbb{R}^{d \times d_e}$  and  $u^0, v^0 \in \mathbb{R}^d$  are learnable parameters.

**Step 2. Graph convolutional layers with MP-GNN.** We apply a series of  $L$  convolution layers, where the node and edge representations are updated with a MP-GNN applied to each graph patch  $G_p = (\mathcal{V}_p, \mathcal{E}_p)$  separately and independently as follows:

$$\begin{aligned} h_{i,p}^{\ell+1} &= f_{\text{node}}(h_{i,p}^\ell, \{h_{j,p}^\ell \mid j \in \mathcal{N}(i)\}, e_{ij,p}^\ell) + g_{\text{patch-n}}(h_p^\ell), \quad h_{i,p}^{\ell+1}, h_{i,p}^\ell, h_p^\ell \in \mathbb{R}^d, \\ e_{ij,p}^{\ell+1} &= f_{\text{edge}}(h_{i,p}^\ell, h_{i,p}^\ell, e_{ij,p}^\ell) + g_{\text{patch-e}}(e_p^\ell), \quad e_{ij,p}^{\ell+1}, e_{ij,p}^\ell, e_p^\ell \in \mathbb{R}^d, \end{aligned} \quad (2)$$

where  $\ell$  is the layer index,  $p$  is the patch index,  $i, j$  denotes the nodes,  $\mathcal{N}(i)$  is the neighborhood of the node  $i$  and functions  $f_{\text{node}}$  and  $f_{\text{edge}}$  (with learnable parameters) define any arbitrary MP-GNN architecture s.a. [16, 18, 74, 37],  $h_p^\ell = \frac{1}{|\mathcal{V}_p|} \sum_{i \in \mathcal{V}_p} h_{i,p}^\ell \in \mathbb{R}^d$ ,  $e_p^\ell = \frac{1}{|\mathcal{E}_p|} \sum_{ij \in \mathcal{E}_p} e_{ij,p}^\ell \in \mathbb{R}^d$  are respectively the mean representations of the patch nodes and patch edges, and  $g_{\text{patch-n}}$ ,  $g_{\text{patch-e}}$  are MLP-based functions that act on  $h_p^\ell$  and  $e_p^\ell$ . For each node and edge covered by more than one patch due to the patch overlapping to include all edges cut by METIS, we update the node/edge representation by averaging over the overlapping patches:

$$h_{i,p}^{l+1} \leftarrow \text{Mean}_{\{k \mid i \in \mathcal{V}_k\}} h_{i,k}^{l+1} \in \mathbb{R}^d, \quad (3)$$

$$e_{ij,p}^{l+1} \leftarrow \text{Mean}_{\{k \mid ij \in \mathcal{E}_k\}} e_{ij,k}^{l+1} \in \mathbb{R}^d, \quad (4)$$

where  $\{k \mid i \in \mathcal{V}_k\}$ ,  $\{k \mid ij \in \mathcal{E}_k\}$  are the set of all patches that cover node  $i$ , edge  $ij$  respectively.

**Step 3. Pooling and readout.** The final step produces a fixed-size vector representation by mean pooling all node vectors in  $G_p$  such that  $h_p = \frac{1}{|\mathcal{V}_p|} \sum_{i \in \mathcal{V}_p} h_{i,p}^{\ell=L} \in \mathbb{R}^d$ , and applying a small MLP to get the patch embedding  $x_{G_p} \in \mathbb{R}^d$ .

Observe that the patch encoder is a MP-GNN, and thus has the inherent limitation of poor long-range dependency. Does it affect the Graph MLP-Mixer to capture long-range interactions? The answer is negative because this problem is limited to large graphs. But for small patch graphs, this issue does not really exist (or is negligible). To give a few numbers, the mean number of nodes and the mean diameter for graph patches are around 3.15 and 1.82 respectively for molecular datasets and around 17.20 and 3.07 for image datasets, see Table 6.

### 4.4 Positional Information

Regular grids offer a natural implicit arrangement for the sequence of image patches and for the pixels inside the image patches. However, such ordering of nodes and patches do not exist for general graphs. This lack of positional information reduces the expressivity of the network. Hence, we use two explicit positional encodings (PE); one absolute PE for the patch nodes and one relative PE for the graph patches.

**Node PE.** Input node features in Eq 1 are augmented with  $p_i \in \mathbb{R}^K$ :

$$h_i^0 = T^0 p_i + U^0 \alpha_i + u^0 \in \mathbb{R}^d, \quad (5)$$

where  $T^0 \in \mathbb{R}^{d \times K}$  is a learnable matrix. The benefits of different PEs are dataset dependent. We follow the strategy in [57] that uses random-walk structural encoding (RWSE) [41] for molecular data and Laplacian eigenvectors encodings [36] for image superpixels. Since Laplacian eigenvectors are defined up to sign flips, the sign of the eigenvectors is randomly flipped during training.

**Patch PE.** Relative positional information between the graph patches can be computed from the original graph adjacency matrix  $A \in \mathbb{R}^{N \times N}$  and the clusters  $\{\mathcal{V}_1, \dots, \mathcal{V}_P\}$  extracted by METIS in Section 4.2. Specifically, we capture relative positional information via the coarsened adjacency matrix  $A^P \in \mathbb{R}^{P \times P}$  over the patch graphs:

$$A_{ij}^P = |\mathcal{V}_i \cap \mathcal{V}_j| = \text{Cut}(\mathcal{V}_i, \mathcal{V}_j), \quad (6)$$

where  $\text{Cut}(\mathcal{V}_i, \mathcal{V}_j) = \sum_{k \in \mathcal{V}_i} \sum_{l \in \mathcal{V}_j} A_{kl}$  is the standard graph cut operator which counts the number of connecting edges between cluster  $\mathcal{V}_i$  and cluster  $\mathcal{V}_j$ . We observe that matrix  $A^P$  is sparse as it only connects patches that are neighbors on the original graph. This can cause poor long-distance interactions. To avoid this situation, we can simply smooth out the adjacency matrix  $A^P$  with any graph diffusion process. In this work, we select the  $n$ -step random walk diffusion process:

$$A_D^P = (D^{-1}A^P)^n \in \mathbb{R}^{P \times P}. \quad (7)$$

#### 4.5 Mixer Layer

For images, the original mixer layer [60] is a simple network that alternates channel and token mixing steps. The token mixing step is performed over the token dimension, while the channel mixing step is carried out over the channel dimension. These two interleaved steps enable information fusion among tokens and channels. The simplicity of the mixer layer has been of great importance to understand that the self-attention mechanism in ViT is not the only critical component to get good performance on visual classification tasks. This has also led to a significant reduction in computational cost with little or no sacrifice in performance. Indeed, the self-attention mechanism in ViT requires  $O(P^2)$  memory and  $O(P^2)$  computation, while the mixer layer in MLP-Mixer needs  $O(P)$  memory and  $O(P)$  computation.

We modify the original mixer layer to introduce positional information between graph tokens. Let  $X \in \mathbb{R}^{P \times d}$  be the patch embedding  $\{x_{G_1}, \dots, x_{G_P}\}$ . The graph mixer layer can be expressed as

$$\begin{aligned} U &= X + (W_2 \sigma(W_1 \text{LayerNorm}(A_D^P X))) \in \mathbb{R}^{P \times d} && \text{Token mixer,} \\ Y &= U + (W_4 \sigma(W_3 \text{LayerNorm}(U)^T))^T \in \mathbb{R}^{P \times d} && \text{Channel mixer,} \end{aligned} \quad (8)$$

where  $A_D^P \in \mathbb{R}^{P \times P}$  is the patch PE from Eq.7,  $\sigma$  is a GELU nonlinearity [75],  $\text{LayerNorm}(\cdot)$  is layer normalization [76], and matrices  $W_1 \in \mathbb{R}^{d_s \times P}$ ,  $W_2 \in \mathbb{R}^{P \times d_s}$ ,  $W_3 \in \mathbb{R}^{d_c \times d}$ ,  $W_4 \in \mathbb{R}^{d \times d_c}$ , where  $d_s$  and  $d_c$  are the tunable hidden widths in the token-mixing and channel-mixing MLPs.

Then we generate the final graph-level representation by mean pooling all the non-empty patches:

$$h_G = \sum_p m_p \cdot x_{G_p} / \sum_p m_p \in \mathbb{R}^d, \quad (9)$$

where  $m_p$  is a binary variable with value 1 for non-empty patches and value 0 for empty patches (since graphs have variable sizes, small graphs can produce empty patches). Finally, we apply a small MLP to get the graph-level target:

$$y_G = \text{MLP}(h_G) \in \mathbb{R} \text{ (regression)} \quad \text{or} \quad \mathbb{R}^{n_c} \text{ (classification)}. \quad (10)$$

#### 4.6 Data augmentation

MLP-Mixer architectures are known to be strong over-fitters [62]. In order to reduce this effect, we propose to introduce some perturbations in METIS as follows. Let  $G = (\mathcal{V}, \mathcal{E})$  be the original graph and  $G' = (\mathcal{V}, \mathcal{E}')$  be the graph after randomly dropping a small set of edges. At each epoch, we apply METIS graph partition algorithm on  $G'$  to get slightly different node partitions  $\{\mathcal{V}_1, \dots, \mathcal{V}_P\}$ . Then, we extract the graph patches  $\{G_1, \dots, G_P\}$  where  $G_i = (\mathcal{V}_i, \mathcal{E}_i)$  is the induced subgraph of the original graph  $G$ , and not the modified  $G'$ . This way, we can produce distinct graph patches at each epoch that retain all the nodes and edges from the original graph. It is worth noting that the drop edge technique we use here is different to the standard data augmentation techniques such as DropEdge [77], and G-Mixup [78], which either add slightly modified copies of existing data or generate synthetic based on existing data. Our dropping edge mechanism is different and actually specific to the Graph MLP-Mixer model.

### 5 Experiments

**Graph Benchmark Datasets.** We evaluate our Graph MLP-Mixer on a wide range of graph benchmarks; 1) **Simulated datasets:** CSL, EXP, SR25 and TreeNeighbourMatch dataset, 2) **Small real-world datasets:** ZINC, MNIST and CIFAR10 from Benchmarking GNNs [36], and MolTOX21 and MolHIV from Open Graph Benchmark (OGB) [65] and 3) **Large real-world datasets:** Peptides-func and Peptides-struct from Long Range Graph Benchmark (LRGB) [66]. Summary statistics of datasets are reported in Appendix A. Experimental details are provided in Appendix B.

Table 2: Comparison with base MP-GNNs. Results are averaged over 4 runs with 4 different seeds.

Model	ZINC	MNIST	CIFAR10	MolTOX21	MolHIV	Peptide-func	Peptide-struct
	MAE ↓	Accuracy ↑	Accuracy ↑	ROCAUC ↑	ROCAUC ↑	Avg. Precision ↑	MAE ↓
GCN	0.1952 ± 0.0057	0.9269 ± 0.0023	0.5423 ± 0.0056	0.7525 ± 0.0031	0.7813 ± 0.0081	0.6328 ± 0.0086	0.2758 ± 0.0012
GCN-MLP-Mixer	<b>0.1323 ± 0.0026</b>	<b>0.9523 ± 0.0016</b>	<b>0.5988 ± 0.0058</b>	<b>0.7829 ± 0.0058</b>	<b>0.7912 ± 0.0121</b>	<b>0.6810 ± 0.0067</b>	<b>0.2492 ± 0.0011</b>
GatedGCN	0.1577 ± 0.0046	0.9776 ± 0.0017	0.6628 ± 0.0017	0.7641 ± 0.0057	0.7874 ± 0.0119	0.6300 ± 0.0029	0.2778 ± 0.0017
GatedGCN-MLP-Mixer	<b>0.1249 ± 0.0020</b>	<b>0.9835 ± 0.0010</b>	<b>0.7082 ± 0.0051</b>	<b>0.7861 ± 0.0048</b>	<b>0.8073 ± 0.0037</b>	<b>0.6856 ± 0.0044</b>	<b>0.2484 ± 0.0016</b>
GINE	0.1072 ± 0.0037	0.9705 ± 0.0023	0.6131 ± 0.0035	0.7730 ± 0.0064	0.7885 ± 0.0034	0.6405 ± 0.0077	0.2780 ± 0.0021
GINE-MLP-Mixer	<b>0.0745 ± 0.0014</b>	<b>0.9801 ± 0.0006</b>	<b>0.6756 ± 0.0086</b>	<b>0.7866 ± 0.0022</b>	<b>0.7951 ± 0.0077</b>	<b>0.6921 ± 0.0054</b>	<b>0.2485 ± 0.0004</b>
GraphTrans	0.1230 ± 0.0018	0.9782 ± 0.0012	0.6809 ± 0.0020	0.7646 ± 0.0055	0.7884 ± 0.0104	0.6313 ± 0.0039	0.2777 ± 0.0025
GraphTrans-MLP-Mixer	<b>0.0798 ± 0.0032</b>	<b>0.9798 ± 0.0008</b>	<b>0.7246 ± 0.0036</b>	<b>0.7874 ± 0.0044</b>	<b>0.7892 ± 0.0116</b>	<b>0.6795 ± 0.0063</b>	<b>0.2475 ± 0.0015</b>

Table 3: Comparison of our best results from Table 2 with the state-of-the-art models (missing values from literature are indicated with '-'). Results are averaged over 4 runs with 4 different seeds.

Model	ZINC	MolHIV	Peptides-func			Peptides-struct		
	MAE ↓	ROCAUC ↑	Avg. Precision ↑	Time/Epoch	Memory	MAE ↓	Time/Epoch	Memory
GT [36]	0.226 ± 0.014	—	—	—	—	—	—	—
GraphiT [53]	0.202 ± 0.011	—	—	—	—	—	—	—
Graphormer [56]	0.122 ± 0.006	—	—	—	—	—	—	—
GPS [57]	<b>0.070 ± 0.004</b>	0.7880 ± 0.0101	0.6562 ± 0.0115	1.3×	6.9×	0.2515 ± 0.0012	1.1×	6.6×
SAN+LapPE [38]	0.139 ± 0.006	0.7775 ± 0.0061	0.6384 ± 0.0121	8.8×	12.4×	0.2683 ± 0.0043	7.4×	11.8×
SAN+RWSE [38]	—	—	0.6439 ± 0.0075	7.5×	19.6×	0.2545 ± 0.0012	6.5×	11.7×
GNN-AK+ [31]	0.080 ± 0.001	0.7961 ± 0.0119	0.6480 ± 0.0089	2.5×	7.8×	0.2736 ± 0.0007	2.1×	7.3×
SUN [32]	0.084 ± 0.002	0.8003 ± 0.0055 <sup>2</sup>	0.6730 ± 0.0078	41.3×	18.8×	0.2498 ± 0.0008	35.7×	16.6×
Graph MLP-Mixer	0.075 ± 0.001	<b>0.8073 ± 0.0037</b>	<b>0.6921 ± 0.0054</b>	1.0×	1.0×	<b>0.2475 ± 0.0015</b>	1.0×	1.0×

### 5.1 Comparison with MP-GNNs

We show in Table 2 that Graph MLP-Mixer lifts the performance of all base MP-GNNs across various datasets, which include GCN [16], GatedGCN [18], GINE [74] and Graph Transformer [37]. We augmented all the base models with the same type of PE as Graph MLP-Mixer to ensure a fair comparison. These promising results demonstrate the generic nature of our proposed architecture which can be applied to any MP-GNNs in practice. Remarkably, Graph MLP-Mixer outperforms the base MP-GNNs by large margins on two LRGB [66] datasets; we can observe an average 0.051 Average Precision improvement on Peptides-func and an average 0.029 MAE decrease on Peptides-struct, which verifies its superiority over MP-GNNs in capturing long-range interaction.

### 5.2 Comparison with SOTAs

Next, we compare Graph MLP-Mixer against popular GNN models with SOTA results, including Graph Transformers (GraphiT, GPS, SAN, etc.) and expressive GNNs (GNN-AK+ and SUN), as shown in Table 3. For small molecular graphs, our model achieved 0.075 on ZINC and 0.8073 on MolHIV. For larger molecular graphs, our model sets new SOTA performance with the best scores of  $0.6921 \pm 0.0054$  on Peptides-func and  $0.2475 \pm 0.0015$  on Peptides-struct.

Besides, Graph MLP-Mixer offers better space-time complexity and scalability. Theoretically, most Graph Transformer models and expressive GNNs, might be computationally infeasible for large graphs, as they need to calculate the full attention and need to run an inner GNN on every node of the graph respectively. Experimentally, we observed that, when training on datasets with hundreds of nodes, SAN+LapPE [55] and SUN [32] require  $7.5\times$  and  $41.3\times$  training time per epoch, and  $12.4\times$  and  $18.8\times$  memory respectively, compared to our model (Table 3 and Table 11).

### 5.3 Graph MLP-Mixer can mitigate over-squashing

TreeNeighboursMatch is a synthetic dataset proposed by Alon and Yahav [44] to simulate the exponentially-growing receptive field in GNNs, which allows us to control the problem radius  $r$ , and thus control the intensity of over-squashing. Figure 2 shows that standard MP-GNNs (i.e., GCN, GGCN, GAT and GIN) fail to generalize on the dataset starting from  $r = 4$  because of over-squashing that squeezes too much information coming from the exponential growth of the tree reception field. However, our model is able to mitigate over-squashing and generalize well until  $r = 7$ , since it

<sup>2</sup>For SUN, we run the code provided by the authors and obtain  $0.7886 \pm 0.0081$  on MolHIV with our 4 seeds.



transmits the long-distance information directly with the mixer layers, instead of applying multiple times of MP and incurring a large-volume receptive field.

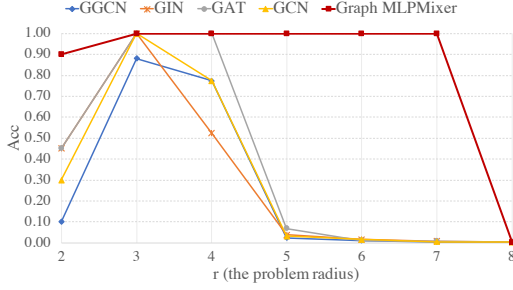


Figure 2: Test Accuracy across problem radius (tree depth) in the NEIGHBORMATCH problem.

Table 4: Empirical evaluation of the expressive power on simulation datasets. Results are averaged over 4 runs with 4 different seeds.

Model	CSL (ACC)	EXP (ACC)	SR25 (ACC)
GCN	10.00 ± 0.00	51.90 ± 1.96	6.67 ± 0.00
GatedGCN	10.00 ± 0.00	51.73 ± 1.65	6.67 ± 0.00
GINE	10.00 ± 0.00	50.69 ± 1.39	6.67 ± 0.00
GraphTrans	10.00 ± 0.00	52.35 ± 2.32	6.67 ± 0.00
GCN-MLP-Mixer	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
GatedGCN-MLP-Mixer	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
GINE-MLP-Mixer	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
GraphTrans-MLP-Mixer	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00

#### 5.4 Graph MLP-Mixer can achieve high expressivity

Although graph PEs s.a. Laplacian eigenvectors [43] or k-step Random Walk PE [40, 41] cannot guarantee two graphs are generally isomorphic, it was shown in [41] that they can distinguish non-isomorphic graphs for which the 1-WL test fails. As a consequence, Graph MLP-Mixer is strictly more powerful than 1-WL. We experimentally validate on three simulation datasets that our model is strictly more powerful than 1&2-WL, and is not less powerful than 3-WL, see Table 4. CSL [34] contains 150 4-regular graphs (1-WL failed) divided into 10 isomorphism classes. EXP [79] containing 600 pairs of non-isomorphic graphs (1-WL/2-WL failed). SR25 [67] has 15 strongly regular graphs (3-WL failed) with 25 nodes each. Our model achieves perfect accuracy on all 3 simulation datasets while MP-GNNs fail.

#### 5.5 Ablation Studies

We evaluate various choices we made to implement each components of the architecture in Appendix C. Table 8 shows how many benefits the METIS can provide against random graph partitioning. Figure 4 evaluates the effect of number of graph patches. Figure 5 studies the effect of patch overlapping with  $k$ -hop extension. Figure 6 shows the effects of two kinds of positional encoding (i.e., node PE and patch PE). Table 9 studies the effect of data augmentation and the trade off between performance and efficiency. Table 10 replaces the Mixer layer in Graph MLP-Mixer with the standard Transformer layer as in ViT [59], keeping the rest the same.

### 6 Complexity Analysis

For each graph  $G = (\mathcal{V}, \mathcal{E})$ , with  $N = |\mathcal{V}|$  being the number of nodes and  $E = |\mathcal{E}|$  being the number of edges, the METIS patch extraction takes  $O(E)$  runtime complexity, and outputs graph patches  $\{G_1, \dots, G_P\}$ , with  $P$  being the pre-defined number of patches. Accordingly, we denote each graph patch as  $G_p = (\mathcal{V}_p, \mathcal{E}_p)$ , with  $N_p = |\mathcal{V}_p|$  being the number of nodes and  $E_p = |\mathcal{E}_p|$  being the number of edges in  $G_p$ . After our one-hop overlapping adjustment, the total number of nodes and edges of all the patches are  $N' = \sum_p N_p \leq PN$  and  $E' = \sum_p E_p \leq PE$ , respectively. Assuming base GNN has  $O(N + E)$  runtime and memory complexity, our patch embedding module has  $O(N' + E')$  runtime and memory complexity, introducing a constant overhead over the base GNN model. For the mixer layers, the complexity is  $O(P)$  as discussed in Section 4.5.

### 7 Conclusion

In this work, we proposed a novel GNN model to improve standard MP-GNN limitations, particularly their low expressivity power and poor long-range dependency, and presented promising results on several benchmark graph datasets. Future work will focus on further exploring graph network architectures with the inductive biases of graph tokens and vision Transformer-like architectures in order to solve fundamental node and link prediction tasks, and possibly without the need of specialized GNN libraries like PyG [80] or DGL [81] by replacing sparse linear algebra operations on graph tokens with dense operations.

#### Acknowledgments

XB is supported by NUS-R-252-000-B97-133 and A\*STAR Grant ID A20H4g2141.

## References

- [1] Federico Monti, Michael Bronstein, and Xavier Bresson. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in neural information processing systems*, 30, 2017.
- [2] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.
- [3] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.
- [4] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pages 1263–1272. PMLR, 2017.
- [5] Miles D Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks. *arXiv preprint arXiv:1909.05862*, 2019.
- [6] Victor Bapst, Thomas Keck, A Grabska-Barwińska, Craig Donner, Ekin Dogus Cubuk, Samuel S Schoenholz, Annette Obika, Alexander WR Nelson, Trevor Back, Demis Hassabis, et al. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16(4):448–454, 2020.
- [7] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, 2021.
- [8] Kai Han, Yunhe Wang, Jianyuan Guo, Yehui Tang, and Enhua Wu. Vision gnn: An image is worth graph of nodes. *arXiv preprint arXiv:2206.00272*, 2022.
- [9] Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, and Bo Long. Graph neural networks for natural language processing: A survey. *arXiv preprint arXiv:2106.06090*, 2021.
- [10] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.
- [11] Jonathan M Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702, 2020.
- [12] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. Utilising graph machine learning within drug discovery and development. *arXiv preprint arXiv:2012.05716*, 2020.
- [13] Yang Li, Buyue Qian, Xianli Zhang, and Hui Liu. Graph neural network-based diagnosis prediction. *Big Data*, 8(5):379–390, 2020.
- [14] Xiaoxiao Li, Yuan Zhou, Nicha Dvornek, Muhan Zhang, Siyuan Gao, Juntang Zhuang, Dustin Scheinost, Lawrence H Staib, Pamela Ventola, and James S Duncan. Brainngn: Interpretable brain graph neural network for fmri analysis. *Medical Image Analysis*, 74:102233, 2021.
- [15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.
- [16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.
- [17] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.
- [18] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.
- [19] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.
- [20] Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI Series*, 2(9):12–16, 1968.
- [21] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

- [22] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.
- [23] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.
- [24] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019.
- [25] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 2019.
- [26] Waiss Azizian and Marc Lelarge. Expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*, 2020.
- [27] Giorgos Bouritsas, Fabrizio Frasca, Stefanos P Zafeiriou, and Michael Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [28] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *Advances in Neural Information Processing Systems*, 33:8017–8029, 2020.
- [29] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Lio, Guido F Montufar, and Michael Bronstein. Weisfeiler and leman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021.
- [30] Muhan Zhang and Pan Li. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34:15734–15747, 2021.
- [31] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness. In *International Conference on Learning Representations*, 2021.
- [32] Fabrizio Frasca, Beatrice Bevilacqua, Michael M Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries. *arXiv preprint arXiv:2206.11140*, 2022.
- [33] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020.
- [34] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In *International Conference on Machine Learning*, pages 4663–4673. PMLR, 2019.
- [35] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020.
- [36] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.
- [37] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. In *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.
- [38] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.
- [39] Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*, 2022.
- [40] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 33, 2020.
- [41] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2021.
- [42] Markus Zopf. 1-wl expressiveness is (almost) all you need. *arXiv preprint arXiv:2202.10156*, 2022.
- [43] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.
- [44] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.
- [45] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

- [46] Andreea Deac, Marc Lackenby, and Petar Veličković. Expander graph propagation. In *The First Learning on Graphs Conference*, 2022.
- [47] Adrián Arnaiz-Rodríguez, Ahmed Begga, Francisco Escolano, and Nuria M Oliver. Diffwire: Inductive graph rewiring via the lovász bound. In *The First Learning on Graphs Conference*, 2022.
- [48] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [49] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [52] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Nee-lakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [53] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.
- [54] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems*, 34:13266–13279, 2021.
- [55] Dexiong Chen, Leslie O’Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022.
- [56] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.
- [57] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.
- [58] Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*, 2022.
- [59] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [60] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021.
- [61] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, et al. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.
- [62] Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. Pay attention to mlps. *Advances in Neural Information Processing Systems*, 34:9204–9215, 2021.
- [63] Ziyu Wang, Wenhao Jiang, Yiming M Zhu, Li Yuan, Yibing Song, and Wei Liu. Dynamixer: a vision mlp architecture with dynamic mixing. In *International Conference on Machine Learning*, pages 22691–22701. PMLR, 2022.
- [64] Weihao Yu, Mi Luo, Pan Zhou, Chenyang Si, Yichen Zhou, Xinchao Wang, Jiashi Feng, and Shuicheng Yan. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10819–10829, 2022.
- [65] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.
- [66] Vijay Prakash Dwivedi, Ladislav Rampášek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *arXiv preprint arXiv:2206.08164*, 2022.

- [67] Muhammet Balcilar, Pierre H  roux, Benoit Gauzere, Pascal Vasseur, S  bastien Adam, and Paul Honeine. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pages 599–608. PMLR, 2021.
- [68] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 702–703, 2020.
- [69] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- [70] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver J. Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. *CoRR*, abs/2006.06830, 2020.
- [71] Aydin Bulu  , Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *Algorithm engineering*, pages 117–158, 2016.
- [72] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.
- [73] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.
- [74] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.
- [75] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.
- [76] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [77] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.
- [78] Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. *arXiv preprint arXiv:2202.07179*, 2022.
- [79] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.
- [80] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.
- [81] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 739–748, 2020.
- [82] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.
- [83] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019.
- [84] Greg Landrum et al. Rdkit: Open-source cheminformatics. 2006, 2006.
- [85] Sandeep Singh, Kumardeep Chaudhary, Sandeep Kumar Dhanda, Sherry Bhalla, Salman Sadullah Usmani, Ankur Gautam, Abhishek Tuknait, Piyush Agrawal, Deepika Mathur, and Gajendra PS Raghava. Satpdb: a database of structurally annotated therapeutic peptides. *Nucleic acids research*, 44(D1):D1119–D1126, 2016.
- [86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [87] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [88] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. *arXiv preprint arXiv:2102.11600*, 2021.
- [89] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How powerful are k-hop message passing graph neural networks. *arXiv preprint arXiv:2205.13328*, 2022.

Table 5: Summary statistics of datasets used in this study

Dataset	#Graphs	#Nodes	Avg. #Nodes	Avg. #Edges	Task	Metric
CSL	150	41	41	164	10-class classif.	Accuracy
EXP	1,200	32-64	44.4	110.2	2-class classif.	Accuracy
SR25	15	25	25	300	15-class classif.	Accuracy
ZINC	12,000	9-37	23.2	24.9	regression	MAE
MNIST	70,000	40-75	70.6	684.4	10-class classif.	Accuracy
CIFAR10	60,000	85-150	117.6	1129.7	10-class classif.	Accuracy
MolTOX21	7,831	1-132	18.57	38.6	12-task classif.	ROCAUC
MolHIV	41,127	2-222	25.5	54.9	binary classif.	ROCAUC
Peptides-func	15,535	8-444	150.9	307.3	10-class classif.	Avg. Precision
Peptides-struct	15,535	8-444	150.9	307.3	regression	MAE
TreeNeighbourMatch (r=2)	96	7	7	6	4-class classif.	Accuracy
TreeNeighbourMatch (r=3)	32,000	15	15	14	8-class classif.	Accuracy
TreeNeighbourMatch (r=4)	64,000	31	31	30	16-class classif.	Accuracy
TreeNeighbourMatch (r=5)	128,000	63	63	62	32-class classif.	Accuracy
TreeNeighbourMatch (r=6)	256,000	127	127	126	64-class classif.	Accuracy
TreeNeighbourMatch (r=7)	512,000	255	255	254	128-class classif.	Accuracy
TreeNeighbourMatch (r=8)	640,000	511	511	510	256-class classif.	Accuracy

## A Datasets Description

We evaluate our Graph MLP-Mixer on a wide range of graph benchmarks. See summary statistics of datasets in Table 5.

**CSL** [34] is a synthetic dataset to test the expressivity of GNNs. CSL has 150 graphs divided into 10 isomorphism classes. Each CSL graph is a 4-regular graph with edges connected to form a cycle and containing skip-links between nodes. The goal of the task is to classify them into corresponding isomorphism classes.

**EXP** [79] contains 600 pairs of 1&2-WL failed graphs. The goal is to map these graphs into two classes.

**SR25** [67] has 15 strongly regular graphs (3-WL failed) with 25 nodes each. SR25 is translated to a 15 way classification problem with the goal of mapping each graph into a different class.

**ZINC** [36] is a subset (12K) of molecular graphs (250K) from a free database of commercially-available compounds [82]. These molecular graphs are between 9 and 37 nodes large. Each node represents a heavy atom (28 possible atom types) and each edge represents a bond (3 possible types). The task is to regress a molecular property known as the constrained solubility. The dataset comes with a predefined 10K/1K/1K train/validation/test split.

**MNIST and CIFAR10** [36] are derived from classical image classification datasets by constructing an 8 nearest-neighbor graph of SLIC superpixels for each image. The resultant graphs are of sizes 40-75 nodes for MNIST and 85-150 nodes for CIFAR10. The 10-class classification tasks and standard dataset splits follow the original image classification datasets, i.e., for MNIST 55K/5K/10K and for CIFAR10 45K/5K/10K train/validation/test graphs. These datasets are sanity-checks, as we expect most GNNs to perform close to 100% for MNIST and well enough for CIFAR10.

**MolTOX21 and MolHIV** [65] are molecular property prediction datasets adopted from the MoleculeNet [83]. All the molecules are pre-processed using RDKit [84]. Each graph represents a molecule, where nodes are atoms, and edges are chemical bonds. Input node features are 9-dimensional, containing atomic number and chirality, as well as other additional atom features such as formal charge and whether the atom is in the ring or not. The datasets come with a predefined scaffold splits based on their two-dimensional structural frameworks, i.e. for MolTOX21 6K/0.78K/0.78K and for MolHIV 32K/4K/4K train/validation/test.

**Peptides-func and Peptides-struct** [66] are derived from 15,535 peptides with a total of 2.3 million nodes retrieved from SAT-Pdb [85]. Both datasets use the same set of graphs but differ in their prediction tasks. These graphs are constructed in such a way that requires long-range interactions (LRI) reasoning to achieve strong performance in a given task. In concrete terms, they are larger graphs: on average 150.94 nodes per graph, and on average 56.99 graph diameter. Thus, they are better suited to benchmarking of graph Transformers or other expressive GNNs that are intended to capture LRI.

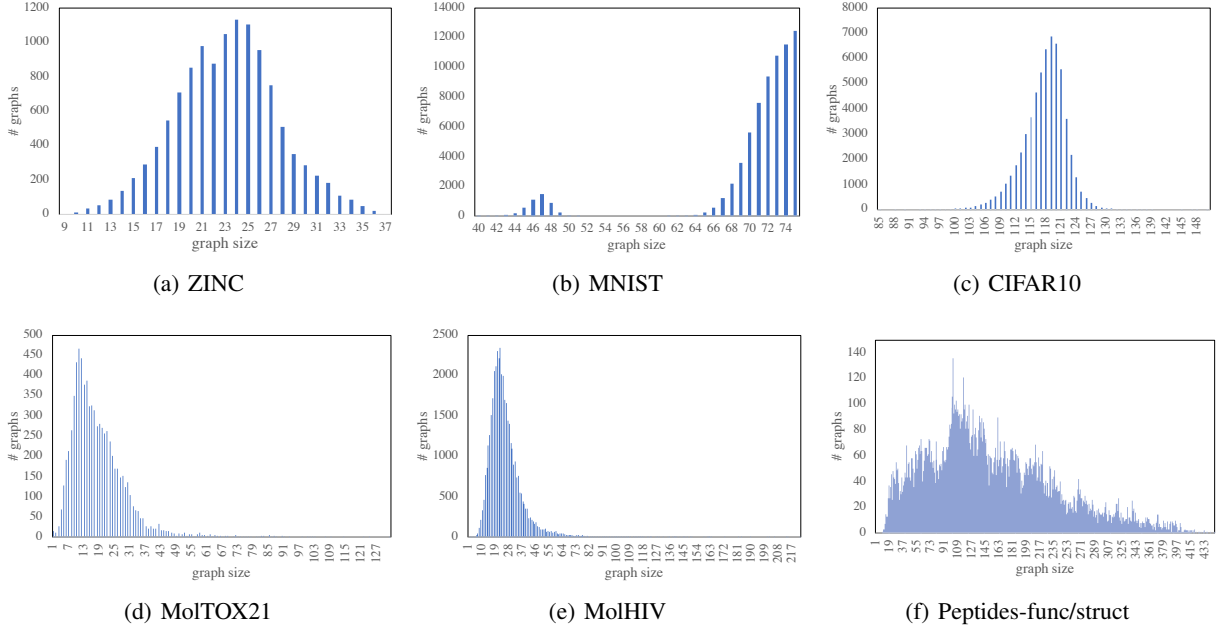


Figure 3: Distributions of the graph sizes.

**TreeNeighbourMatch** is a synthetic dataset proposed by Alon and Yahav [44] to highlight the inherent problem of over-squashing in GNNs. It is designed to simulate the exponentially-growing receptive field while allowing us to control the problem radius  $r$ , and thus control the intensity of over-squashing. Specifically, each graph is a binary tree of depth  $depth$  (a.k.a. problem radius  $r$ ). The goal is to predict a label for the target node, where the correct answer lies in one of the leaf nodes. Therefore, the TreeNeighbourMatch problem requires information to be propagated from all leaf nodes to the target node before predicting the label, causing the issue of over-squashing at the target node.

**Distributions of the graph sizes.** We plot of the distributions of the graph sizes (*i.e.*, the number of nodes in each data sample) of these datasets in Figure 3.

**Patch Size and Diameter.** We set the number of patches to 32 by default. Summary statistics of graph patches are presented in Table 6.

## B Experiment Details

We implement our model using PyTorch [86] and PyG [80]. We ran our experiments on NVIDIA RTX A5000 GPUs. We run each experiment with 4 different seeds, reporting the averaged results at the epoch achieving the best validation metric. For optimization, we use Adam [87] optimizer, with the default settings of  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$ , and  $\epsilon = 1e^{-8}$ . We observe large fluctuations in the validation metric with the common Adam optimizer on the OGB datasets (*i.e.*, MolHIV and MolTOX21), as also observed in [32, 30, 25]. We consider following the practice of SUN [32] by employing the ASAM optimizer [88] to reduce such fluctuations. We use the same hyperparameter with batch size of 32 and learning rate of 0.01 without further tuning.

**Simulation Datasets.** For CSL and EXP, we run the 5-fold cross validation with stratified sampling to ensure class distribution remains the same across the splits [36, 30]. For SR25 dataset, we follow the evaluation process in [31, 89] that directly train and validate the model on the whole dataset and report the best performance.

**Real-World Datasets.** For benchmarking datasets from Dwivedi et al. [36], we followed the most commonly used parameter budgets: up to 500k parameters for ZINC; For MolTOX21 and MolHIV from OGB [65], there is no upper limit on the number of parameters. For peptides-func and peptides-struct from LRGB [66], we followed the parameter budget  $\sim 500k$ . All real world evaluated benchmarks define a standard train/validation/test dataset split.

**Baselines.** We use GCN [16], GatedGCN [18], GINE [74] and Graph Transformer [37] as our baseline models, which also serve as the base patch encoder of Graph MLP-Mixer. The hidden size is set to 128 and the number of layers is set to 4 by default. For TreeNeighbourMatch datasets, we follow the experimental protocol introduced in [44], that is, for

Table 6: Summary statistics of graph patches.

Dataset	# Patch	# Node			Diameter		
		Mean	Min	Max	Mean	Min	Max
CSL	32	5.80	5	8	2.28	2	3
EXP	32	4.07	2	11	2.31	1	5
SR25	32	13.00	13	13	2.00	2	2
ZINC	32	3.15	2	7	1.82	1	3
MNIST	32	14.36	9	28	2.85	2	5
CIFAR10	32	17.20	10	35	3.07	2	7
MolTOX21	32	3.15	1	10	1.80	0	6
MolHIV	32	3.27	1	13	1.87	0	8
Peptides-func	32	7.08	1	20	4.15	0	14
Peptides-struct	32	7.08	1	20	4.15	0	14
TreeNeighbourMatch(r=2)	8	1.86	1	3	0.86	0	2
TreeNeighbourMatch(r=3)	32	1.93	1	3	0.93	0	2
TreeNeighbourMatch(r=4)	32	1.97	1	3	0.97	0	2
TreeNeighbourMatch(r=5)	32	3.28	1	5	2.25	0	3
TreeNeighbourMatch(r=6)	32	5.34	3	8	3.31	2	5
TreeNeighbourMatch(r=7)	32	9.19	7	14	4.33	4	5
TreeNeighbourMatch(r=8)	32	17.03	15	23	6.17	6	8

TreeNeighbourMatch dataset with problem radius  $r = \text{depth}$ , we implemented a network with  $r + 1$  graph layers to allow an additional nonlinearity after the information from the leaves reaches the target node.

**Graph MLP-Mixer.** The hidden size is set to 128, and the number of GNN layers and Mixer layers is set to 4. Except that for LRGB datasets, we reduce the number of Mixer layers to 2 to fulfill the parameter budget  $\sim 500k$ .

**SOTA models.** In Table 3 and Table 11, results are referenced directly from literature if available, otherwise are reproduced using authors’ official code. To enable a fair comparison of speed/memory complexity (Table 11), we set the batch size to 128 all the SOTA models and ours and reduce the batch size by half if OOM until the model and batch data can be fit into the memory. Besides, all experiments are run on the same machine.

**Positional Encodings.** As the most appropriate choice of node positional encoding (NodePE) is dataset and task dependent, we follow the practice of Rampásek et al. [57], Dwivedi et al. [66], see Table 7. We have already augmented all the base models (GCN, GatedGCN, GINE and GraphTrans) in Table 2 with the same type of NodePE as Graph MLP-Mixer to ensure a fair comparison.

Table 7: Summary statistics of node positional encoding (NodePE).

Dataset	CSL	EXP	SR25	ZINC	MNIST	CIFAR10	MolTOX21	MolHIV	Peptides-fun	Peptides-struct
NodePE	RWSE-8	RWSE-8	LapPE-8	RWSE-20	LapPE-8	LapPE-8	–	–	RWSE-16	RWSE-16

## C Detailed Ablation Studies

### C.1 Effect of Graph Partition Algorithm

We select METIS [73] as the graph partitioning algorithm, and provide the ablation study for how many benefits the METIS can provide against random graph partitioning. For random graph partition, nodes are randomly assigned to a pre-defined number of patches. We apply data augmentation as described in Section 4.6 to both algorithms. Table 8 shows that using METIS as the graph partition algorithm consistently gives better performance than random node partition, especially on graphs with more nodes and edges (such as Peptides-func), which corresponds to our intuition that nodes and edges composing a patch should share similar semantic or information. Nevertheless, it is interesting to see that random graph partitioning is still able to achieve reasonable results, which shows that the performance of the model is not solely supported by the quality of the patches.



Table 8: Comparison of METIS and random graph partition algorithm.

Model	ZINC (MAE ↓)		Peptides-struct (MAE ↓)	
	METIS	Random	METIS	Random
GCN-MLP-Mixer	0.1323 ± 0.0026	0.1391 ± 0.0021	0.2492 ± 0.0011	0.2550 ± 0.0022
GatedGCN-MLP-Mixer	0.1249 ± 0.0020	0.1264 ± 0.0029	0.2484 ± 0.0016	0.2575 ± 0.0012
GINE-MLP-Mixer	0.0745 ± 0.0014	0.0766 ± 0.0019	0.2485 ± 0.0004	0.2526 ± 0.0015
GraphTrans-MLP-Mixer	0.0798 ± 0.0032	0.0837 ± 0.0043	0.2475 ± 0.0015	0.2607 ± 0.0016

## C.2 Effect of #Patches

We observe in Figure 4 when increasing the number of graph patches, performance increases first and then flattens out (with small fluctuations) when #Patch=24. We set the number of patches to 32 by default.

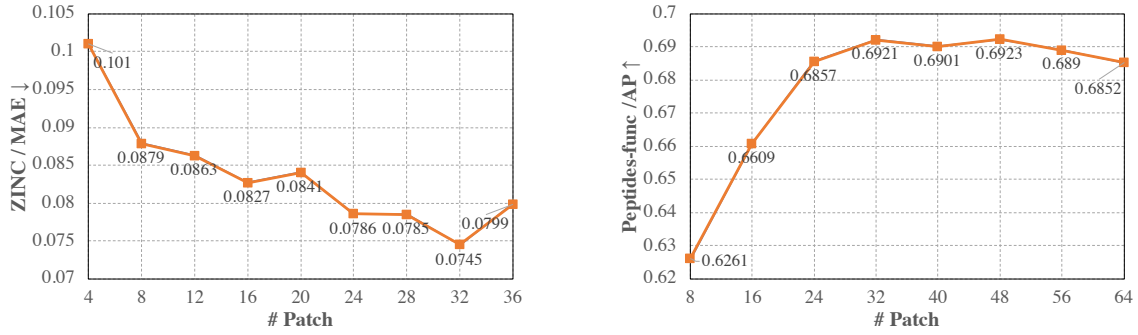


Figure 4: Effect of the number of patches.

## C.3 Effect of Patch Overlapping

In Figure 5, we observe a clear performance increase when graph patches are overlapping with each other (0-hop vs 1-hop), which is consistent with our intuition that extracting non-overlapping patches implies losing important edge information. We further expand graph patches to their  $k$ -hop neighbourhood. Performance increases first and then flattens out or begins to decrease when  $k = 2$  for ZINC and  $k = 3$  for Peptides-func. We set  $k = 1$  by default.

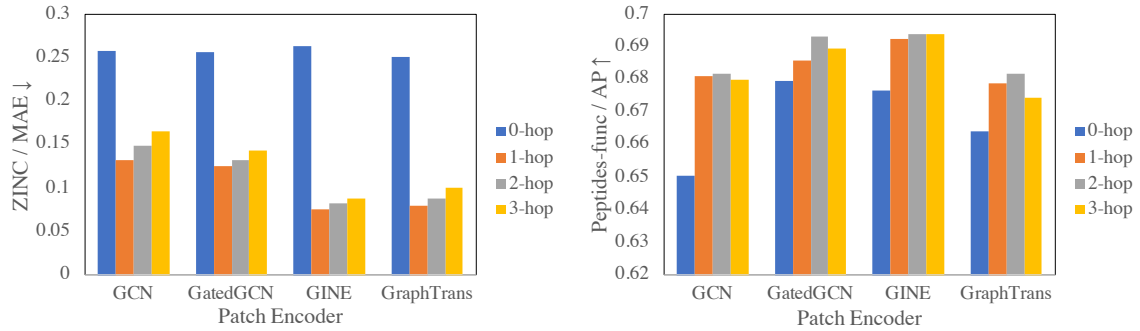


Figure 5: Effect of the patch overlapping with k-hop extension.

## C.4 Effect of Positional Encoding

It was proved in [34, 35] that unique and permutation-invariant positional encoding (PE) increases the representation power of any MP-GNN, i.e. PE leads to GNNs strictly more powerful than the 1-WL test. PE is thus important from a



Table 10: Mixer Layer vs. Transformer Layer.

Model	ZINC (MAE ↓)		Peptides-struct (MAE ↓)	
	Graph MLP-Mixer	Graph ViT	Graph MLP-Mixer	Graph ViT
GCN-	0.1347 ± 0.0057	0.1640 ± 0.0029	0.2493 ± 0.0016	0.2567 ± 0.0020
GatedGCN-	0.1240 ± 0.0095	0.1609 ± 0.0033	0.2464 ± 0.0019	0.2562 ± 0.0017
GINE-	0.0804 ± 0.0050	0.1238 ± 0.0039	0.2468 ± 0.0015	0.2562 ± 0.0015
GraphTrans-	0.0791 ± 0.0022	0.1337 ± 0.0156	0.2480 ± 0.0017	0.2563 ± 0.0012

### C.7 Long Range Graph Benchmark

We have provided additional experiments with the recent Long Range Graph Benchmark (LRGB) [66] to demonstrate that Graph MLP-Mixer is able to capture long-range interactions. In LRGB, Peptides-func and Peptides-struct are two graph-level prediction datasets, consisting of 15,535 graphs with a total of 2.3 million nodes. The graphs are one order of magnitude larger than ZINC, MolTOX21 and MolHIV with 151 nodes per graph on average and a mean graph diameter of 57. As such, they are better suited to evaluate models enabled with long-range dependencies, as they contain larger graphs and more data points. The performance is reported in Table 2, Table 3 and in Table 11.

We summarize the main results as follows: 1) Graph MLP-Mixer sets new SOTA performance with the best scores of  $0.6921 \pm 0.0054$  on Peptides-func and  $0.2475 \pm 0.0015$  on Peptides-struct (Table 3), demonstrating the ability of the model to better capture long-range relationships. 2) Compared with MP-GNNs (Table 2), Graph MLP-Mixer significantly outperforms the base MP-GNNs; we can observe an average 0.051 Average Precision improvement on Peptides-func and an average 0.029 MAE decrease on Peptides-struct, which verifies its superiority over MP-GNNs in capturing long-range interaction. 3) Graph MLP-Mixer provides significantly better speed/memory complexity compared to Graph Transformer and expressive gnn models, especially when training with large graphs, such as SAN+LSPE [55] and SUN [32]. For example, SUN gives similar performance to Graph MLP-Mixer, 0.6730 on Peptides-func and 0.2498 on Peptides-struct, but requires 40x memory and 19x training time (Table 11).

Table 11: Comparison of our best results from Table 2 with the state-of-the-art Models on large real world datasets [66].

Model	# Params	Peptide-func			Peptide-struct		
		Avg. Precision ↑	Time (S/Epoch)	Memory (MB)	MAE ↓	Time (S/Epoch)	Memory (MB)
GCN	508k	0.5930 ± 0.0023	4.59	696	0.3496 ± 0.0013	4.51	686
GINE	476k	0.5498 ± 0.0079	3.94	659	0.3547 ± 0.0045	3.84	658
GatedGCN	509k	0.5864 ± 0.0077	5.48	1,038	0.3420 ± 0.0013	5.31	1,029
GatedGCN + RWSE	506k	0.6069 ± 0.0035	5.75	1,035	0.3357 ± 0.0006	5.61	1,038
Transformer + LapPE	488k	0.6326 ± 0.0126	9.74 (1.1×)	6,661 (6.6×)	0.2529 ± 0.0016	9.61 (0.9×)	6,646 (6.4×)
SAN + LapPE [55]	493k	0.6384 ± 0.0121	80.47 (8.8×)	12,493 (12.4×)	0.2683 ± 0.0043	79.41 (7.4×)	12,226 (11.8×)
SAN + RWSE [55]	500k	0.6439 ± 0.0075	68.44 (7.5×)	19,691 (19.6×)	0.2545 ± 0.0012	70.39 (6.5×)	12,111 (11.7×)
GPS [57]	504k	0.6562 ± 0.0115	11.83 (1.3×)	6,904 (6.9×)	0.2515 ± 0.0012	11.74 (1.1×)	6,878 (6.6×)
GNN-AK+ [31]	631k	0.6480 ± 0.0089	22.52 (2.5×)	7,855 (7.8×)	0.2736 ± 0.0007	22.11 (2.1×)	7,634 (7.3×)
SUN [32]	508k	0.6730 ± 0.0078	376.66 (41.3×)	18,941 (18.8×)	0.2498 ± 0.0008	384.26 (35.7×)	17,215 (16.6×)
GCN-MLP-Mixer	328k	0.6810 ± 0.0067	8.91	716	0.2492 ± 0.0011	9.09	667
GatedGCN-MLP-Mixer	526k	0.6856 ± 0.0044	9.49	922	0.2484 ± 0.0016	9.22	889
GINE-MLP-Mixer	395k	<b>0.6921 ± 0.0054</b>	9.13 (1.0×)	1,006 (1.0×)	0.2485 ± 0.0004	8.91	969
GraphTrans-MLP-Mixer	591k	0.6795 ± 0.0063	11.07	1,063	<b>0.2475 ± 0.0015</b>	10.76 (1.0×)	1,039 (1.0×)