A Generalization of ViT/MLP-Mixer to Graphs

Xiaoxin He [1] **Bryan Hooi** [1 2] **Thomas Laurent** [3] **Adam Perold** [4] **Yann LeCun** [5 6] **Xavier Bresson** [1]

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 ViT/MLP-Mixer, that holds three key properties. First, they capture long-range dependency and mitigate the issue of over-squashing as demonstrated on Long Range Graph Benchmark and 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. The source code is available for reproducibility at: https://github.com/XiaoxinHe/Graph-ViT-MLPMixer.

1. Message-Passing GNNs and the Limitations

In this first section, we present the background of the project by introducing the standard Message-Passing (MP) GNNs

[1]School of Computing, University of Singapore [2]Institute of Data Science, National University of Singapore [3]Loyola Marymount University [4]Element, Inc. [5]New York University [6]Meta AI. Correspondence to: Xiaoxin He <he.xiaoxin@u.nus.edu>.

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 (Monti et al., 2017; Berg et al., 2017), chemistry (Duvenaud et al., 2015; Gilmer et al., 2017), physics (Cranmer et al., 2019; Bapst et al., 2020), transportation (Derrow-Pinion et al., 2021), vision (Han et al., 2022a), natural language processing (NLP) (Wu et al., 2021a), knowledge graphs (Schlichtkrull et al., 2018), drug design (Stokes et al., 2020; Gaudelet et al., 2020) and medical domain (Li et al., 2020b; 2021). Most GNNs are designed to have two core components. First, a structural message-passing mechanism s.a. Defferrard et al. (2016); Kipf & Welling (2017); Hamilton et al. (2017); Monti et al. (2017); Bresson & Laurent (2017); Gilmer et al. (2017); Veličković et al. (2018) 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 (Weisfeiler & Leman, 1968), as first proposed in Xu et al. (2019); Morris et al. (2019). Later on, Maron et al. (2018) introduced a general class of k-order WL-GNNs that can be proved to universally represent any class of k-WL graphs (Maron et al., 2019; Chen et al., 2019). 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 (Maron et al., 2019; Chen et al., 2019; Azizian & Lelarge, 2020), 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. In order to reduce memory and speed complexi-

ties of WL-GNNs while keeping high expressivity, several works have focused on designing graph networks from their sub-structures s.a. sub-graph isomorphism (Bouritsas et al., 2022), sub-graph routing mechanism (Alsentzer et al., 2020), cellular WL sub-graphs (Bodnar et al., 2021), and k-hop egonet sub-graphs (Xu et al., 2019; Zhang & Li, 2021; Chen et al., 2019; Zhao et al., 2021; Frasca et al., 2022).

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 (Chen et al., 2020). 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. (2019); Loukas (2020) 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 (Murphy et al., 2019), Laplacian eigenvectors (Dwivedi et al., 2020; Dwivedi & Bresson, 2021; Kreuzer et al., 2021; Lim et al., 2022) and k-step Random Walk (Li et al., 2020a; Dwivedi et al., 2021). All these graph PEs lead to GNNs strictly more powerful than the 1-WL test, which seems to be enough expressivity in practice (Zopf, 2022). 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 (Belkin & Niyogi, 2003) 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 limitations 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 (Alon & Yahav, 2020; Topping et al., 2022). 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. Deac et al. (2022); Arnaiz-Rodríguez et al. (2022). Over-

squashing is well-known since recurrent neural networks (Hochreiter & Schmidhuber, 1997), which have led to the development of the (self- and cross-)attention mechanisms for the translation task (Bahdanau et al., 2014; Vaswani et al., 2017) first, and then for more general NLP tasks (Devlin et al., 2018; Brown et al., 2020). 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.
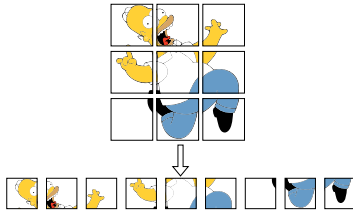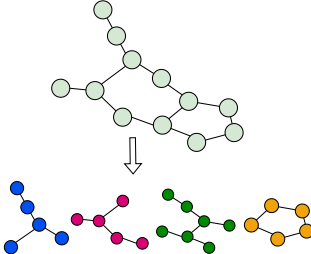
Graph Transformers. Dwivedi & Bresson (2021) generalize Transformers to graphs, with graph Laplacian eigenvectors as node PE, and incorporating graph structure into the permutation-invariant attention function. SAN and LSPE (Kreuzer et al., 2021; Dwivedi et al., 2021) further improve with PE learned from Laplacian and random walk operators. GraphiT (Mialon et al., 2021) encodes relative PE derived from diffusion kernels into the attention mechanism. GraphTrans (Wu et al., 2021b) add Transformers on the top of standard GNN layers. SAT (Chen et al., 2022a) proposes a novel self-attention mechanism that incorporates structural information into the standard self-attention module by using a GNN to compute subgraph representations. Graphormer (Ying et al., 2021) introduce three structural encodings, with great success on large molecular benchmarks. GPS (Rampášek et al., 2022) categorizes the different types of PE and puts forward a hybrid MPNN+Transformer architecture. We refer to Min et al. (2022) 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. A detailed description of related literature can be found in Appendix A.

2. Generalizing ViT/MLP-Mixer to Graphs

In the following, we explain the importance of generalizing the ViT/MLP-Mixer architectures from CV to graphs.

ViT and MLP-Mixer in computer vision. Transformers have gained remarkable success in CV and NLP, most notably with architectures like ViT (Dosovitskiy et al., 2020) and BERT (Devlin et al., 2018). The success of transformers has been long attributed to the attention mechanism (Vaswani et al., 2017), 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

Table 1. Differences between ViT/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
ViT / MLP-Mixer	MLP / Channel mixer MHA / Token mixer	MLP / Channel mixer gMHA / Token mixer

Tolstikhin et al. (2021) 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 (Tolstikhin et al., 2021); 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. These plain networks (Tolstikhin et al., 2021; Touvron et al., 2021; Liu et al., 2021; Wang et al., 2022) 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 (Yu et al., 2022).

The benefits of generalizing ViT/MLP-Mixer from grids to graphs. Standard MP-GNNs have linear learning/inference complexities but low representation power and poor long-range dependency. Graph Transformers address these two problems but loose the computational effi-

ciency with a quadratic complexity price. A generalization of ViT/MLP-Mixer to graphs overcomes the computational bottleneck of Graph Transformers and solves the issue of long-distance dependency, hence going beyond standard MP-GNNs. However, achieving such a successful generalization is challenging given the irregular and variable nature of graphs. See Section 3 for a detailed presentation of theses challenges.

Contribution. Our contributions are listed as follows.

- We identify the key challenges to generalize ViT/MLP-Mixer from images to graphs and design a new efficient class of GNNs, namely Graph ViT/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 (Dwivedi et al., 2020) and the Open Graph Benchmark (OGB) (Hu et al., 2020); specifically, with 0.073 MAE on ZINC and 0.7997 ROCAUC on MolHIV.

- We demonstrate the capacity of the proposed model to capture long-range dependencies with SOTA

performance on Long Range Graph Benchmark (LRGB) (Dwivedi et al., 2022) while keeping low complexity, to mitigate the over-squashing issue on the TreeNeighbourMatch dataset (Alon & Yahav, 2020), and to reach the 3-WL expressive power on the SR25 dataset (Balcilar et al., 2021).

- 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

We list the main questions when adapting MLP-Mixer from images to graphs in the following and 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 topology 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.

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

(4) How to reduce over-fitting for Graph ViT/MLP-Mixer? ViT/MLP-Mixer architectures are known to be strong over-fitters (Liu et al., 2021). Most MLP-variants (Tolstikhin et al., 2021; Touvron et al., 2021; Wang et al., 2022) 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 (Cubuk et al., 2020), mixup (Zhang et al., 2017), 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 (Zhao et al., 2020). 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 $\{\mathcal{V}_1, ..., \mathcal{V}_P\}$ 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 representation and y_G be the graph-level target.

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 $x_p \in$
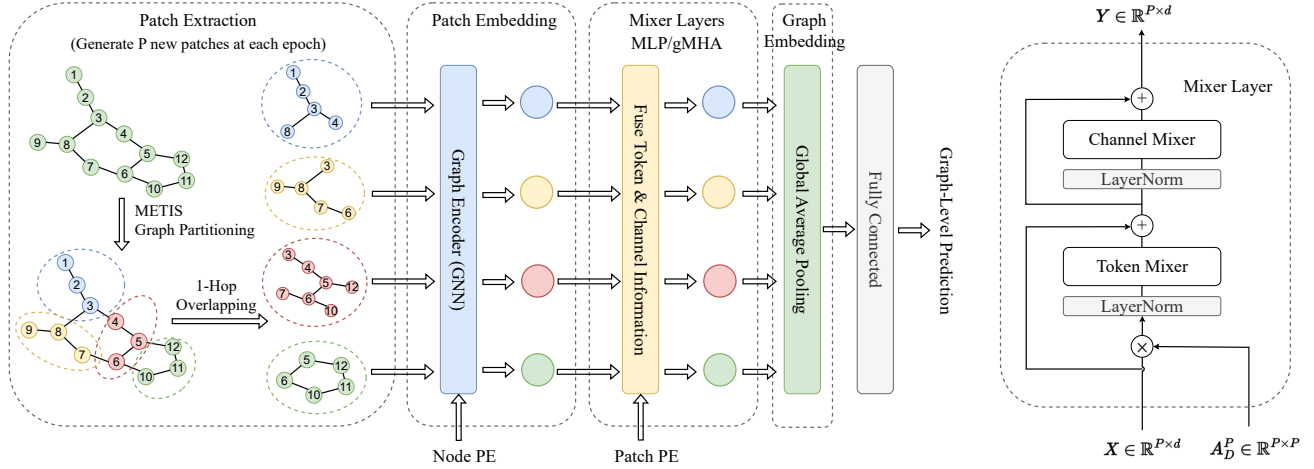
Figure 1. The basic architecture of the proposed Graph ViT/MLP-Mixer. They consist of a patch extraction module, a patch embedding module, a sequence of mixer layers, a global average pooling, and a classifier head. The patch extraction module partitions graphs into overlapping patches. The patch embedding 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, MLP or graph-based multi-head attention (gMHA), 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).

$\mathbb{R}^{P \times (R^2 C)}$, where (R, R) is the resolution of each image patch, and $P = HW/R^2$ is the resulting number of patches.

Unlike images with fixed resolution, extracting graph patches is more challenging, see Table 1. 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. $O(E)$.

METIS (Karypis & Kumar, 1998) is a graph clustering algorithm with one of the best trade-off accuracy and speed. It partitions a graph into a pre-defined number of clusters 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 it as our 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: $\{\mathcal{V}_1, ..., \mathcal{V}_P\}$ such that $\mathcal{V} = \mathcal{V}_1 \cup ... \cup \mathcal{V}_P$ 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.

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} 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:

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

where $h_i^0 \in \mathbb{R}^d$, $e_{ij}^0 \in \mathbb{R}^d$, $U^0 \in \mathbb{R}^{d \times d_n}$, $V^0 \in \mathbb{R}^{d \times d_e}$ 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 $G_p = (\mathcal{V}_p, \mathcal{E}_p)$ as follows:

$$
\begin{aligned}
h_{i,p}^{\ell+1} &= f_{\text{node}}(h_{i,p}^\ell, \{h_{j,p}^\ell | j \in \mathcal{N}(i)\}, e_{ij,p}^\ell) + g_{\text{patch-node}}(h_p^\ell), \\
e_{ij,p}^{\ell+1} &= f_{\text{edge}}(h_{i,p}^\ell, h_{i,p}^\ell, e_{ij,p}^\ell) + g_{\text{patch-edge}}(e_p^\ell),
\end{aligned}
\tag{2}
$$

where $h_{i,p}^{\ell+1}, h_{i,p}^\ell, h_p^\ell, e_{ij,p}^{\ell+1}, e_{ij,p}^\ell, e_p^\ell \in \mathbb{R}^d$, ℓ 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_{node} and f_{edge} (with learnable parameters) define any arbitrary MP-GNN architecture s.a. (Kipf & Welling, 2017; Bresson & Laurent, 2017; Hu et al., 2019; Dwivedi & Bresson, 2021), $h_p^\ell = \frac{1}{|\mathcal{V}_p|} \sum_{i \in \mathcal{V}_p} h_{i,p}^l \in \mathbb{R}^d$, $e_p^\ell = \frac{1}{|\mathcal{E}_p|} \sum_{ij \in \mathcal{E}_p} e_{ij,p}^l \in \mathbb{R}^d$ are respectively the mean representations of the patch nodes and patch edges, and $g_{\text{patch-node}}$, $g_{\text{patch-edge}}$ are MLP-based functions that act on h_p^ℓ and e_p^ℓ. 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:

$$h_{i,p}^{l+1} \leftarrow \underset{\{k | i \in \mathcal{V}_k\}}{\text{Mean}} h_{i,k}^{l+1}; \quad e_{ij,p}^{l+1} \leftarrow \underset{\{k | ij \in \mathcal{E}_k\}}{\text{Mean}} e_{ij,k}^{l+1}, \tag{3}$$

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

Step 3. Pooling and readout. The final step is 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 fixed-sized 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 7.

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 $p_i \in \mathbb{R}^K$, with a learnable matrix $T^0 \in \mathbb{R}^{d \times K}$:

$$h_i^0 = T^0 p_i + U^0 \alpha_i + u^0 \in \mathbb{R}^d, \tag{4}$$

The benefits of different PEs are dataset dependent. We follow the strategy in (Rampášek et al., 2022) that uses random-walk structural encoding (RWSE) (Dwivedi et al., 2021) for molecular data and Laplacian eigenvectors encodings (Dwivedi et al., 2020) 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 $\{\mathcal{V}_1, ..., \mathcal{V}_P\}$ 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:

$$A_{ij}^P = |\mathcal{V}_i \cap \mathcal{V}_j| = \text{Cut}(\mathcal{V}_i, \mathcal{V}_j), \tag{5}$$

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 extract the positional encoding $\hat{p}_i \in \mathbb{R}^{\hat{K}}$ at the patch level, similar to the node level, which will be injected (after a linear transformation) into the first layer of the mixer block:

$$x_i^0 = \hat{T}^0 \hat{p}_i + \hat{U}^0 x_i + \hat{u}^0 \in \mathbb{R}^d, \tag{6}$$

where x_i is the patch embedding.

4.5. Mixer Layer

For images, the original mixer layer (Tolstikhin et al., 2021) 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 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.

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 ↑	AP ↑	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	**0.1347 ± 0.0020**	0.9516 ± 0.0027	0.6111 ± 0.0017	0.7816 ± 0.0075	**0.7929 ± 0.0111**	0.6832 ± 0.0061	0.2486 ± 0.0041
GCN-ViT	0.1688 ± 0.0095	**0.9600 ± 0.0015**	**0.6367 ± 0.0027**	0.7820 ± 0.0096	0.7780 ± 0.0120	**0.6855 ± 0.0049**	**0.2468 ± 0.0015**
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	0.1244 ± 0.0053	0.9832 ± 0.0004	0.7060 ± 0.0022	**0.7910 ± 0.0040**	**0.7976 ± 0.0136**	0.6932 ± 0.0017	0.2508 ± 0.0007
GatedGCN-ViT	**0.1421 ± 0.0031**	**0.9846 ± 0.0009**	**0.7158 ± 0.0009**	0.7857 ± 0.0028	0.7734 ± 0.0114	**0.6942 ± 0.0075**	**0.2465 ± 0.0015**
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	**0.0733 ± 0.0014**	0.9809 ± 0.0004	0.6833 ± 0.0022	**0.7868 ± 0.0043**	**0.7997 ± 0.0102**	**0.6970 ± 0.0080**	0.2494 ± 0.0007
GINE-ViT	0.0849 ± 0.0047	**0.9820 ± 0.0005**	**0.6967 ± 0.0040**	0.7851 ± 0.0077	0.7792 ± 0.0149	0.6919 ± 0.0085	**0.2449 ± 0.0016**
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	**0.0773 ± 0.0030**	**0.9742 ± 0.0011**	**0.7396 ± 0.0033**	0.7817 ± 0.0040	**0.7969 ± 0.0061**	0.6858 ± 0.0062	0.2480 ± 0.0013
GraphTrans-ViT	0.0960 ± 0.0073	0.9725 ± 0.0023	0.7211 ± 0.0055	**0.7835 ± 0.0032**	0.7755 ± 0.0208	**0.6876 ± 0.0059**	**0.2455 ± 0.0027**

Let $X \in \mathbb{R}^{P \times d}$ be the patch embedding $\{x_{G_1}, ..., x_{G_P}\}$, the graph mixer layer can be expressed as

$$U = X + (W_2 \sigma(W_1 \text{ LayerNorm}(X))) \in \mathbb{R}^{P \times d},$$
$$Y = U + (W_4 \sigma(W_3 \text{LayerNorm}(U)^T))^T \in \mathbb{R}^{P \times d}, \quad (7)$$

where σ is a GELU nonlinearity (Hendrycks & Gimpel, 2016), LayerNorm($\cdot$) is layer normalization (Ba et al., 2016), 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.

Alternatively, we can formulate a graph transformer layer to incorporate the self-attention mechanism as in ViT:

$$U = X + \text{gMHA}(\text{LayerNorm}(X)) \in \mathbb{R}^{P \times d},$$
$$Y = U + \text{MLP}(\text{LayerNorm}(U)) \in \mathbb{R}^{P \times d}, \quad (8)$$

where gMHA (graph-based multi-head attention) is designed to capture token dependencies based on the given graph topology. In Eq.(8), gHMA is defined as $(A^P \odot \text{softmax}(\frac{QK^T}{\sqrt{d}}))V$ but other options are possible to characterize the gHMA mechanism, as studied in Appendix F.

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:

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

4.6. Data augmentation

MLP-Mixer architectures are known to be strong overfitters (Liu et al., 2021). 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 $\{\mathcal{V}_1, ..., \mathcal{V}_P\}$. Then, we extract the graph patches $\{G_1, ..., G_P\}$ 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.

5. Experiments

Graph Benchmark Datasets. We evaluate our Graph ViT/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 (Dwivedi et al., 2020), and MolTOX21 and MolHIV from OGB (Hu et al., 2020) and 3) **Large real-world datasets:** Peptides-func and Peptides-struct from LRGB (Dwivedi et al., 2022). Details are provided in Appendix B and Appendix C.

5.1. Comparison with MP-GNNs

We show in Table 2 that ViT/Graph MLP-Mixer lifts the performance of all base MP-GNNs across various datasets, which include GCN (Kipf & Welling, 2017), GatedGCN (Bresson & Laurent, 2017), GINE (Hu et al., 2019) and Graph Transformer (Dwivedi & Bresson, 2021). 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 ViT/MLP-Mixer outperforms the base MP-GNNs by large margins on two LRGB (Dwivedi et al., 2022) datasets; we can observe an average 0.056 Average Precision improvement on Peptides-func and an average 0.028 MAE decrease on Peptides-struct, which verifies its superiority over MP-GNNs in capturing

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-strcut		
	MAE ↓	ROCAUC ↑	AP ↑	Time	Mem.	MAE ↓	Time	Mem.
GT (Dwivedi et al., 2020)	0.226 ± 0.014	–	–	–	–	–	–	–
GraphiT (Mialon et al., 2021)	0.202 ± 0.011	–	–	–	–	–	–	–
Graphormer (Ying et al., 2021)	0.122 ± 0.006	–	–	–	–	–	–	–
GPS (Rampášek et al., 2022)	**0.070 ± 0.004**	0.7880 ± 0.0101	0.6562 ± 0.0115	1.4×	6.8×	0.2515 ± 0.0012	1.3×	8.3×
SAN+LapPE (Kreuzer et al., 2021)	0.139 ± 0.006	0.7775 ± 0.0061	0.6384 ± 0.0121	9.4×	12.4×	0.2683 ± 0.0043	8.8×	14.7×
SAN+RWSE (Kreuzer et al., 2021)	–	–	0.6439 ± 0.0075	8.0×	19.5×	0.2545 ± 0.0012	7.9×	14.5×
GNN-AK+ (Zhao et al., 2021)	0.080 ± 0.001	0.7961 ± 0.0119	0.6480 ± 0.0089	2.6×	7.8×	0.2736 ± 0.0007	2.5×	9.2×
SUN (Frasca et al., 2022)	0.084 ± 0.002	0.8003 ± 0.0055[1]	0.6730 ± 0.0078	43.8×	18.8×	0.2498 ± 0.0008	42.7×	20.7×
CIN (Bodnar et al., 2021)	0.079 ± 0.006[2]	0.8094 ± 0.0057	–	–	–	–	–	–
Graph MLP-Mixer	0.073 ± 0.001	**0.7997 ± 0.0102**	**0.6970 ± 0.0080**	1.0×	1.0×	0.2475 ± 0.0015	1.0×	1.2×
Graph ViT	0.085 ± 0.005	0.7792 ± 0.0149	0.6942 ± 0.0075	1.1×	0.8×	**0.2449 ± 0.0016**	1.0×	1.0×

long-range interaction.

5.2. Comparison with SOTAs

Next, we compare Graph ViT/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 0.073 on ZINC and 0.7997 on MolHIV. For larger molecular graphs, our model sets new SOTA performance with the best scores of 0.6970 on Peptides-func and 0.2449 on Peptides-struct.

Besides, Graph ViT/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 (Chen et al., 2022a) and SUN (Frasca et al., 2022) require 9.4× and 43.8× training time per epoch, and 12.4× and 18.8× memory respectively, compared to our model (Table 3 and Table 17).

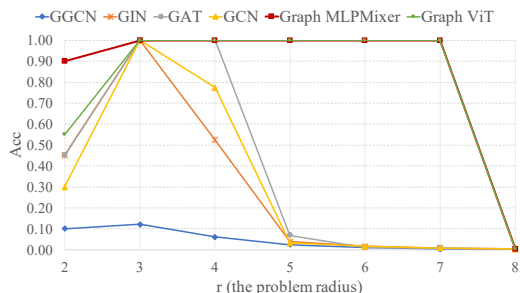## 5.3. Graph ViT/MLP-Mixer can mitigate over-squashing

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

TreeNeighbourMatch is a synthetic dataset proposed by Alon & Yahav (2020) to provide an intuition into over-squashing. Each example is a binary tree of depth r. The goal is to predict an alphabetical label for the target node, where the correct answer is the label of the leaf node that has the same degree as the target node. Figure 2 shows that standard MP-GNNs (i.e., GCN, GGCN, GAT and GIN) fail to generalize on the dataset from $r = 4$, whereas our model mitigates over-squashing and generalizes well until $r = 7$. As for why this happens, Alon & Yahav (2020) show that GNNs fail to solve larger TreeNeighbourMatch cases as they 'squash' information about the graph into the target node's embedding, which can hold a limited amount of information. In contrast, Graph ViT/MLP-Mixer avoids this problem as it transmits long-range information directly without 'squashing.' Concretely, Appendix I shows a simple construction illustrating that our model can solve TreeNeighbourMatch cases while avoiding the inherent limitations of MP-GNNs discussed by Alon & Yahav (2020).

5.4. Graph ViT/MLP-Mixer can achieve empirical high expressivity

Table 4. Empirical evaluation of the expressive power on simulation datasets, averaging 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

We experimentally evaluate the expressive power of Graph

---

[1]For SUN, we run the official code and obtain 0.7886 ± 0.0081 on MolHIV with our 4 seeds.

[2]For CIN, the reporting score is not obtained with the budget of 500k parameters but with 1.7M parameters (3× more) when running their official code.

ViT/MLP-Mixer on three simulated datasets. CSL (Murphy et al., 2019) contains 150 4-regular graphs that cannot be distinguished with a 1-WL isomorphism test. EXP (Abboud et al., 2020) contains 600 pairs of non-isomorphic graphs: both 1-WL and 2-WL tests fail at differentiating these graphs. Finally, SR25 (Balcilar et al., 2021) has 15 strongly regular graphs with 25 nodes each that cannot be discerned with a 3-WL test. Numerical experiments show that our model achieves perfect accuracy on all 3 datasets while MP-GNNs fail, see Table 4. Our results are only empirical. Due to the nonlocal way in which information is passed from one layer to the other, a direct analytical comparison between the proposed neural network and the Weisfeiler-Lehman test is challenging.

5.5. Ablation Studies

In our ablation studies, we evaluated various choices made during the implementation of each component of the architecture. The details of these studies can be found in the appendix. Appendix D focuses on the design of the patch extraction process, including the effects of the graph partition algorithm (Table 9), patch size (Figure 4), patch overlapping (Figure 5), and other related aspects. Appendix E presents the effects of two types of positional encoding, i.e., node PE and patch PE. Appendix G investigates the effect of data augmentation and explores the trade-off between performance and efficiency. In Appendix F, we delve into different designs of the gMHA mechanism in the Graph ViT. Additionally, we provide a complexity analysis in Appendix J and discuss the limitations in Appendix K.

6. Conclusion

In this work, we proposed a novel GNN architecture 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 (Fey & Lenssen, 2019) or DGL (Zheng et al., 2020) by replacing sparse linear algebra operations on graph tokens with dense operations.

Acknowledgments

References

Abboud, R., Ceylan, I. I., Grohe, M., and Lukasiewicz, T. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179*, 2020.

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

Alsentzer, E., Finlayson, S., Li, M., and Zitnik, M. Subgraph neural networks. *Advances in Neural Information Processing Systems*, 33:8017–8029, 2020.

Arnaiz-Rodríguez, A., Begga, A., Escolano, F., and Oliver, N. M. Diffwire: Inductive graph rewiring via the lovász bound. In *The First Learning on Graphs Conference*, 2022.

Azizian, W. and Lelarge, M. Expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*, 2020.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Balcilar, M., Héroux, P., Gauzere, B., Vasseur, P., Adam, S., and Honeine, P. Breaking the limits of message passing graph neural networks. In *International Conference on Machine Learning*, pp. 599–608. PMLR, 2021.

Bapst, V., Keck, T., Grabska-Barwińska, A., Donner, C., Cubuk, E. D., Schoenholz, S. S., Obika, A., Nelson, A. W., Back, T., Hassabis, D., et al. Unveiling the predictive power of static structure in glassy systems. *Nature Physics*, 16(4):448–454, 2020.

Belkin, M. and Niyogi, P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.

Berg, R. v. d., Kipf, T. N., and Welling, M. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

Bodnar, C., Frasca, F., Otter, N., Wang, Y., Lio, P., Montufar, G. F., and Bronstein, M. Weisfeiler and lehman go cellular: Cw networks. *Advances in Neural Information Processing Systems*, 34:2625–2640, 2021.

Bouritsas, G., Frasca, F., Zafeiriou, S. P., and Bronstein, M. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Bresson, X. and Laurent, T. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., and Schulz, C. Recent advances in graph partitioning. *Algorithm engineering*, pp. 117–158, 2016.

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, 2022a.

Chen, J., Gao, K., Li, G., and He, K. Nagphormer: Neighborhood aggregation graph transformer for node classification in large graphs. *arXiv preprint arXiv:2206.04910*, 2022b.

Chen, Z., Chen, L., Villar, S., and Bruna, J. On the equivalence between graph isomorphism testing and function approximation with gnns. *Advances in neural information processing systems*, 2019.

Chen, Z., Chen, L., Villar, S., and Bruna, J. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020.

Chung, F. R. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

Cranmer, M. D., Xu, R., Battaglia, P., and Ho, S. Learning symbolic physics with graph networks. *arXiv preprint arXiv:1909.05862*, 2019.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 702–703, 2020.

Deac, A., Lackenby, M., and Veličković, P. Expander graph propagation. In *The First Learning on Graphs Conference*, 2022.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

Derrow-Pinion, A., She, J., Wong, D., Lange, O., Hester, T., Perez, L., Nunkesser, M., Lee, S., Guo, X., Wiltshire, B., et al. Eta prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 3767–3776, 2021.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. In *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2021.

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.

Feng, J., Chen, Y., Li, F., Sarkar, A., and Zhang, M. How powerful are k-hop message passing graph neural networks. *arXiv preprint arXiv:2205.13328*, 2022.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Frasca, F., Bevilacqua, B., Bronstein, M. M., and Maron, H. Understanding and extending subgraph gnns by rethinking their symmetries. *arXiv preprint arXiv:2206.11140*, 2022.

Freitas, S., Dong, Y., Neil, J., and Chau, D. H. A large-scale database for graph representation learning. *arXiv preprint arXiv:2011.07682*, 2020.

Gaudelet, T., Day, B., Jamasb, A. R., Soman, J., Regep, C., Liu, G., Hayter, J. B., Vickers, R., Roberts, C., Tang, J., et al. Utilising graph machine learning within drug discovery and development. *arXiv preprint arXiv:2012.05716*, 2020.

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.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Han, K., Wang, Y., Guo, J., Tang, Y., and Wu, E. Vision gnn: An image is worth graph of nodes. *arXiv preprint arXiv:2206.00272*, 2022a.

Han, X., Jiang, Z., Liu, N., and Hu, X. G-mixup: Graph data augmentation for graph classification. *arXiv preprint arXiv:2202.07179*, 2022b.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

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.

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

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.

Kuang, W., WANG, Z., Li, Y., Wei, Z., and Ding, B. Coarformer: Transformer for large graph via graph coarsening, 2022. URL https://openreview.net/forum?id=fkjO_FKVzw.

Kwon, J., Kim, J., Park, H., and Choi, I. K. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. *arXiv preprint arXiv:2102.11600*, 2021.

Landrum, G. et al. Rdkit: Open-source cheminformatics. 2006, 2006.

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 33, 2020a.

Li, X., Zhou, Y., Dvornek, N., Zhang, M., Gao, S., Zhuang, J., Scheinost, D., Staib, L. H., Ventola, P., and Duncan, J. S. Braingnn: Interpretable brain graph neural network for fmri analysis. *Medical Image Analysis*, 74:102233, 2021.

Li, Y., Qian, B., Zhang, X., and Liu, H. Graph neural network-based diagnosis prediction. *Big Data*, 8(5):379–390, 2020b.

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, H., Dai, Z., So, D., and Le, Q. V. Pay attention to mlps. *Advances in Neural Information Processing Systems*, 34: 9204–9215, 2021.

Loukas, A. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020.

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

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019.

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

Min, E., Chen, R., Bian, Y., Xu, T., Zhao, K., Huang, W., Zhao, P., Huang, J., Ananiadou, S., and Rong, Y. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*, 2022.

Monti, F., Bronstein, M., and Bresson, X. Geometric matrix completion with recurrent multi-graph neural networks. *Advances in neural information processing systems*, 30, 2017.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Murphy, R., Srinivasan, B., Rao, V., and Ribeiro, B. Relational pooling for graph representations. In *International Conference on Machine Learning*, pp. 4663–4673. PMLR, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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.

Rong, Y., Huang, W., Xu, T., and Huang, J. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. v. d., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., and Sinop, A. K. Exphormer: Sparse transformers for graphs. *arXiv preprint arXiv:2303.06147*, 2023.

Singh, S., Chaudhary, K., Dhanda, S. K., Bhalla, S., Usmani, S. S., Gautam, A., Tuknait, A., Agrawal, P., Mathur, D., and Raghava, G. P. Satpdb: a database of structurally annotated therapeutic peptides. *Nucleic acids research*, 44(D1):D1119–D1126, 2016.

Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z., et al. A deep learning approach to antibiotic discovery. *Cell*, 180(4): 688–702, 2020.

Szklarczyk, D., Gable, A. L., Lyon, D., Junge, A., Wyder, S., Huerta-Cepas, J., Simonovic, M., Doncheva, N. T., Morris, J. H., Bork, P., 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.

Tolstikhin, I. O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in Neural Information Processing Systems*, 34:24261–24272, 2021.

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., and Bronstein, M. M. 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.

Touvron, H., Bojanowski, P., Caron, M., Cord, M., El-Nouby, A., Grave, E., Izacard, G., Joulin, A., Synnaeve, G., Verbeek, J., et al. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 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., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Wang, Z., Jiang, W., Zhu, Y. M., Yuan, L., Song, Y., and Liu, W. Dynamixer: a vision mlp architecture with dynamic mixing. In *International Conference on Machine Learning*, pp. 22691–22701. PMLR, 2022.

Weisfeiler, B. and Leman, A. The reduction of a graph to canonical form and the algebra which appears therein. *NTI Series*, 2(9):12–16, 1968.

Wu, L., Chen, Y., Shen, K., Guo, X., Gao, H., Li, S., Pei, J., and Long, B. Graph neural networks for natural language processing: A survey. *arXiv preprint arXiv:2106.06090*, 2021a.

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, 2021b.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

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.

Yu, W., Luo, M., Zhou, P., Si, C., Zhou, Y., Wang, X., Feng, J., and Yan, S. Metaformer is actually what you need for vision. In *Proceedings of the IEEE/CVF Conference*

on Computer Vision and Pattern Recognition, pp. 10819–10829, 2022.

Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

Zhang, M. and Li, P. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34: 15734–15747, 2021.

Zhang, Z., Liu, Q., Hu, Q., and Lee, C.-K. Hierarchical graph transformer with adaptive node sampling. *arXiv preprint arXiv:2210.03930*, 2022.

Zhao, L., Jin, W., Akoglu, L., and Shah, N. From stars to subgraphs: Uplifting any gnn with local structure awareness. In *International Conference on Learning Representations*, 2021.

Zhao, T., Liu, Y., Neves, L., Woodford, O. J., Jiang, M., and Shah, N. Data augmentation for graph neural networks. *CoRR*, abs/2006.06830, 2020.

Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., Xiong, H., Zhang, Z., and Karypis, G. Dgl-ke: Training knowledge graph embeddings at scale. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 739–748, 2020.

Zopf, M. 1-wl expressiveness is (almost) all you need. *arXiv preprint arXiv:2202.10156*, 2022.

A. Related Work

Table 5. Comparison of different hierarchical graph models.

	GNN	Transformer	Graph Coarsening	Local Info.	Global Info.
Coarformer (Kuang et al., 2022)	✓	✓	✓(non-overlap, static)	GNN on original graph	MHA on coarsen graph
Exphormer (Shirzad et al., 2023)	✓	✓	✗	GNN on original graph	MHA on expander graph
ANS-GT (Zhang et al., 2022)	✗	✓	✓(non-overlap, static)	adaptive node sampling strategy	sampled nodes from the coarsened graph
NAGphormer (Chen et al., 2022b)	✗	✓	✗	MHA on multi-hop neighbour	–
Graph MLP-Mixer (Ours)	✓	✓	✓(overlap, dynamic)	GNN on graph patches	token mixer across patches

We briefly review the hierarchical graph models (Kuang et al., 2022; Shirzad et al., 2023; Zhang et al., 2022; Chen et al., 2022b) and highlight the main differences among them.

Coarformer (Kuang et al., 2022) combines MPNNs and Transformers, using a GNN-based module for local information and a Transformer-based module for global information. Exphormer (Shirzad et al., 2023) also employs MPNN+Transformer, using GNN and Transformer modules on the original graph and expander graph, respectively. ANS-GT (Zhang et al., 2022) introduces a node-sampling-based GT with hierarchical attention and graph coarsening. NAGphormer (Chen et al., 2022b) treats each node as a token sequence and aggregates multi-hop information using attention-based readout.

Main differences: 1) GNN/Transformer module. Coarformer, Exphormer, SAT and ours use a hybrid MPNN+Transformer architecture while ANS-GT and NAGphormer rely solely on Transformers. However, there are notable differences between these approaches as we do not use any Transformer but rather MLP as our backbone. Besides, our MPNN operates on small graph patches instead of the entire graph as Coarformer and Exphormer. Furthermore, SAT and our architecture are sequential, while Coarformer and Exphormer combine MP-GNNs and GT in parallel.

2) Graph coarsening module. Coarformer, ANS-GT and SAT use a graph coarsening mechanism. These methods perform graph coarsening as a pre-processing step and generate static and non-overlapping graph patches. In contrast, we perform graph coarsening with a stochastic version of Metis on-the-fly, generating dynamic and overlapping graph patches.

3) Graph embedding module. The ways to capture local and global information are different as stated in the above reviews and the above summary Table 5.

In summary, although these aforementioned hierarchical graph models share similarities with our model, the major difference between these models and our work is that we do not use Graph Transformer (GT) as the backbone architecture but an alternative architecture based on ViT/MLP-Mixer and graphs. We believe moving from GT to Graph ViT/MLP-Mixer as a new backbone/high-level architecture has the potential to open a new line of work for GNN design (by enhancing the building blocks of the proposed architecture such as graph clustering, graph embedding, mixer layer, positional encoding, (pre-)training, etc).

B. Datasets Description

We evaluate our Graph MLP-Mixer on a wide range of graph benchmarks. See summary statistics of datasets in Table 6.

CSL (Murphy et al., 2019) 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 (Abboud et al., 2020) contains 600 pairs of 1&2-WL failed graphs. The goal is to map these graphs into two classes.

SR25 (Balcilar et al., 2021) 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 (Dwivedi et al., 2020) is a subset (12K) of molecular graphs (250K) from a free database of commercially-available compounds (Irwin et al., 2012). 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 (Dwivedi et al., 2020) 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

Table 6. 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.	Average Precision (AP)
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

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 (Hu et al., 2020) are molecular property prediction datasets adopted from the MoleculeNet (Szklarczyk et al., 2019). All the molecules are pre-processed using RDKit (Landrum et al., 2006). 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 (Dwivedi et al., 2022) are derived from 15,535 peptides with a total of 2.3 million nodes retrieved from SAT-Pdb (Singh et al., 2016). 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.

TreeNeighbourMatch is a synthetic dataset proposed by Alon & Yahav (2020) 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 leave nodes. Therefore, the TreeNeighbourMatch problem requires information to be propagated from all leave 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 7.

C. Experiment Details

We implement out model using PyTorch (Paszke et al., 2019) and PyG (Fey & Lenssen, 2019). 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 (Kingma & Ba, 2014) 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
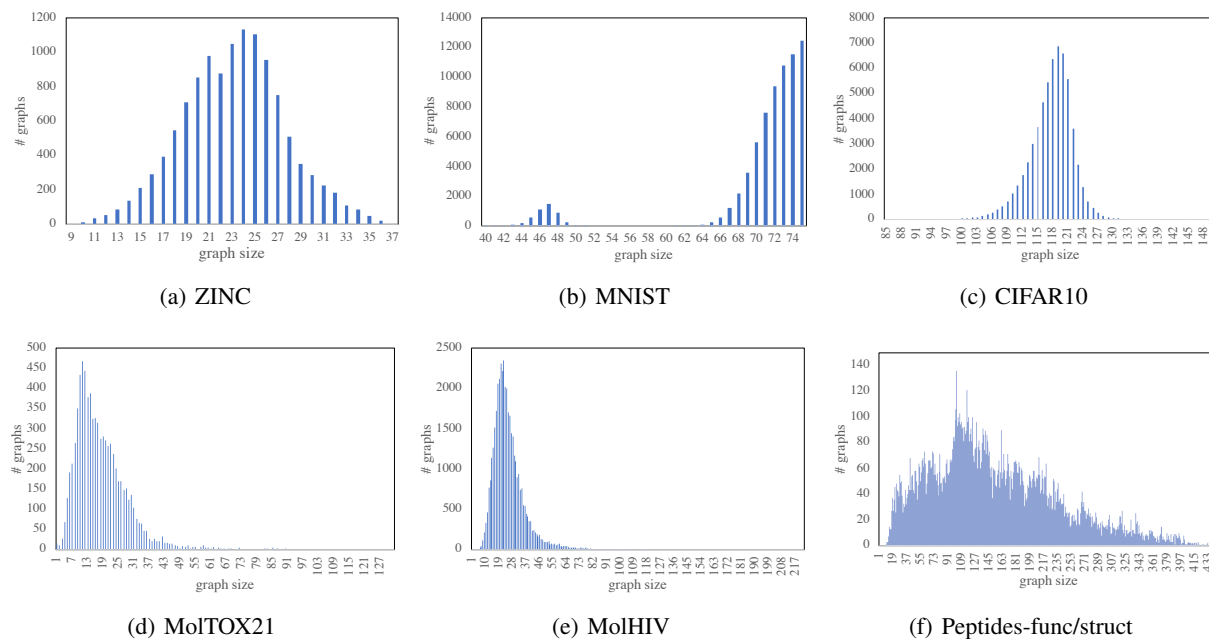
(a) ZINC

(b) MNIST

(c) CIFAR10

(d) MolTOX21

(e) MolHIV

(f) Peptides-func/struct

Figure 3. Distributions of the graph sizes.

Table 7. 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

Adam optimizer on the OGB datasets (i.e., MolHIV and MolTOX21), as also observed in (Frasca et al., 2022; Zhang & Li, 2021; Chen et al., 2019). We consider following the practice of SUN (Frasca et al., 2022) by employing the ASAM optimizer (Kwon et al., 2021) 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 (Dwivedi et al., 2020; Zhang & Li, 2021). For SR25 dataset, we follow the evaluation process in (Zhao et al., 2021; Feng et al., 2022) 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. (2020), we followed the most commonly used parameter budgets: up to 500k parameters for ZINC; For MolTOX21 and MolHIV from OGB (Hu et al., 2020), there is no upper limit on the number of parameters. For peptides-func and peptides-struct from LRGB (Dwivedi et al., 2022), we followed the parameter budget ∼500k. All real world evaluated benchmarks define a standard train/validation/test dataset split.

Baselines. We use GCN (Kipf & Welling, 2017), GatedGCN (Bresson & Laurent, 2017), GINE (Hu et al., 2019) and Graph Transformer (Dwivedi & Bresson, 2021) as our baseline models, which also server 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 (Alon & Yahav, 2020), that is, for TreeNeighbourMatch dataset with problem radias $r = depth$, 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 ∼500k.

SOTA models. In Table 3 and Table 17, 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 17), 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 Rampášek et al. (2022); Dwivedi et al. (2022), see Table 8. 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 8. Summary statistics of positional encoding (PE).

	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
PatchPE	RWSE-8	RWSE-8	RWSE-8	RWSE-8	RWSE-8	RWSE-8	–	–	RWSE-8	RWSE-8

D. Studies on Patch Extraction Module

D.1. Effect of Patch Extraction

Table 9. Effect of patch extraction: ✗means no patch extraction and ✓means uses patch extraction.

Model	Patch Extraction	ZINC (MAE↓)	Peptides-func (AP↑)
GCN-MLP-Mixer	✗	0.2495 ± 0.0040	0.6341 ± 0.0139
	✓	0.1347 ± 0.0020	0.6832 ± 0.0061
GatedGCN-MLP-Mixer	✗	0.2521 ± 0.0084	0.6230 ± 0.0110
	✓	0.1244 ± 0.0053	0.6932 ± 0.0017
GINE-MLP-Mixer	✗	0.2558 ± 0.0059	0.6350 ± 0.0038
	✓	0.0733 ± 0.0014	0.6970 ± 0.0080
GraphTrans-MLP-Mixer	✗	0.2538 ± 0.0067	0.6224 ± 0.0112
	✓	0.0773 ± 0.0030	0.6858 ± 0.0062

We conducted an experiment where we ran the Graph MLP-Mixer without the patch extraction process, treating each individual node as a patch. The results of this experiment are presented in Table 9. The patch extraction process is critical. We believe that the patch extraction process, which includes Metis partition and 1-hop extension, helps to capture important local information about the graph structure.

Table 10. Comparison of METIS and random graph partition algorithm.

Model	ZINC (MAE ↓)		Peptides-struct (MAE ↓)	
	METIS	Random	METIS	Random
GCN-MLP-Mixer	0.1347 ± 0.0020	0.1435 ± 0.0122	0.2486 ± 0.0041	0.2565 ± 0.0031
GatedGCN-MLP-Mixer	0.1244 ± 0.0053	0.1284 ± 0.0074	0.2508 ± 0.0007	0.2539 ± 0.0012
GINE-MLP-Mixer	0.0733 ± 0.0014	0.0708 ± 0.0020	0.2494 ± 0.0007	0.2559 ± 0.0012
GraphTrans-MLP-Mixer	0.0773 ± 0.0030	0.0767 ± 0.0019	0.2480 ± 0.0013	0.2574 ± 0.0025

D.2. Effect of Graph Partition Algorithm

Graph partitioning algorithms have been studied for decades (Buluç et al., 2016) given their importance in identifying meaningful clusters. Mathematically, graph partitioning is known to be NP-hard (Chung, 1997). Approximations are thus required. A graph clustering algorithm with one of the best trade-off accuracy and speed is METIS (Karypis & Kumar, 1998), 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.

We 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 10 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.

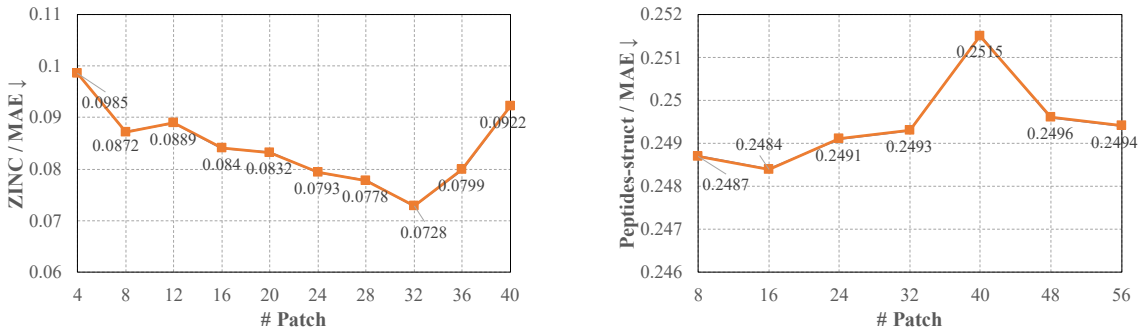### D.3. Effect of Number of Patches

Figure 4. Effect of the number of patches.

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

D.4. 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.
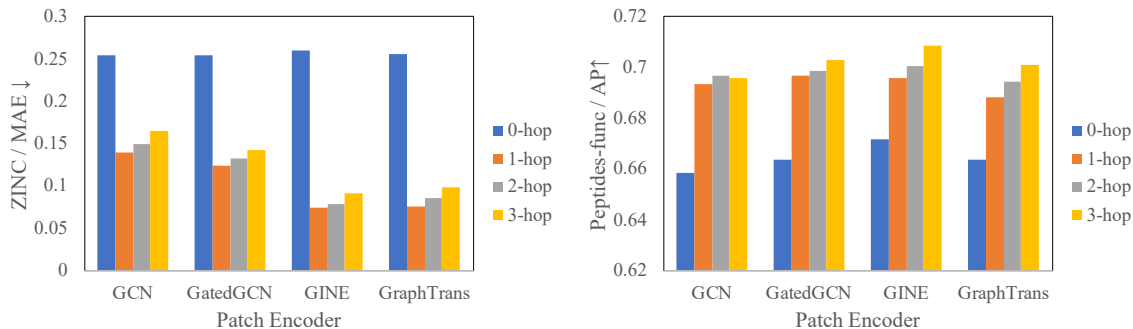
Figure 5. Effect of the patch overlapping with k-hop extension.

Table 11. Accuracy on EXP with different patch sizes P, averaging over 4 runs with 4 different seeds

Model	P=2	P=4	P=8	P=16	P=32
GCN-MLP-Mixer	57.54 ± 3.87	99.44 ± 0.59	99.69 ± 0.98	100.00 ± 0.00	100.00 ± 0.00
GatedGCN-MLP-Mixer	67.65 ± 2.01	99.77 ± 0.37	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
GINE-MLP-Mixer	57.75 ± 3.80	99.58 ± 0.45	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00
GraphTrans-MLP-Mixer	73.79 ± 1.52	96.77 ± 8.43	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00

D.5. Patch Size and WL Expressivity

We evaluated the 2-WL and 3-WL expressivity on the benchmark datasets available to us, which indeed have small graphs. As we do not have access to 2-WL/3-WL datasets with larger graph sizes, we studied the impact of performance with a smaller number of patches in Table 11. As expected, expressivity increases when the number of patches increases as well. Given these experiential results, we also suppose that for larger graphs, we would need to increase the number of patches to maintain expressivity.

E. Studies on Positional Encoding

E.1. Effect of Positional Encoding

Table 12. Effect of positional encoding. We study the effects of node PE and patch PE by removing one of them in turn from our model while keeping the other components unchanged.

Dataset	Method	GCN-MLP-Mixer	Gated-MLP-Mixer	GINE-MLP-Mixer	GraphTrans-MLP-Mixer
ZINC	Full	0.1347 ± 0.0020	0.1244 ± 0.0053	0.0733 ± 0.0014	0.0773 ± 0.0030
	- NodePE	0.1944 ± 0.0061	0.1775 ± 0.0031	0.1225 ± 0.0070	0.1393 ± 0.0122
	- PatchPE	0.1414 ± 0.0058	0.1250 ± 0.0026	0.0746 ± 0.0010	0.0778 ± 0.0029
	- Both	0.2207 ± 0.0072	0.1883 ± 0.0096	0.1160 ± 0.0023	0.1700 ± 0.0064
Peptides-func	Full	0.6832 ± 0.0061	0.6932 ± 0.0017	0.6970 ± 0.0080	0.6858 ± 0.0062
	- NodePE	0.6688 ± 0.0039	0.6864 ± 0.0080	0.6868 ± 0.0034	0.6763 ± 0.0030
	- PatchPE	0.6871 ± 0.0055	0.6934 ± 0.0055	0.6933 ± 0.0104	0.6882 ± 0.0076
	- Both	0.6760 ± 0.0078	0.6847 ± 0.0034	0.6756 ± 0.0070	0.6783 ± 0.0088

It was proved in (Murphy et al., 2019; Loukas, 2020) 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 theoretical point of view but, unfortunately, theory does not provide any guidance on the choice of PE for a given graph dataset and task. Consequently, the choice of PE is so far arbitrary and is selected by trial-and-error experiments such as (Rampášek et al., 2022; Lim et al., 2022) to cite the most recent PE-based GNNs.

Our experiments show that PE may or not be useful, see Table 12. Thus, PE increases the expressivity power of GNNs but not necessarily their generalization performance. In other words, they improve over-fitting but not necessarily generalization.

In conclusion, PE is certainly useful to improve the quality of GNN prediction given the theory and the increased number of published works on this topic, but more mathematical progress is needed to identify more relevant choices and provides consistent result improvement.

E.2. Positional Encoding and Patch Size

Table 13. Ablation with combining effects of PE and patch size on ZINC.

Patch Size	2	4	16	32
Full	0.0983 ± 0.0042	0.1011 ± 0.0103	0.0799 ± 0.0037	0.0743 ± 0.0049
- Node PE	0.1589 ± 0.0056	0.1414 ± 0.0061	0.1307 ± 0.0107	0.1154 ± 0.0032
- Patch PE	0.1081 ± 0.0007	0.1076 ± 0.0110	0.0840 ± 0.0035	0.0744 ± 0.0037
- Both	0.1677 ± 0.0045	0.1532 ± 0.0051	0.1284 ± 0.0018	0.1187 ± 0.0050

Table 14. Ablation with combining effects of PE and patch size on Peptide-func.

Patch Size	2	4	16	32	64
Full	0.6578 ± 0.0063	0.6675 ± 0.0037	0.6855 ± 0.0039	0.6939 ± 0.0034	0.6944 ± 0.0074
- Node PE	0.6613 ± 0.0063	0.6708 ± 0.0065	0.6864 ± 0.0069	0.6873 ± 0.0033	0.6789 ± 0.0047
- Patch PE	0.6594 ± 0.0059	0.6724 ± 0.0051	0.6937 ± 0.0068	0.6939 ± 0.0062	0.6865 ± 0.0061
- Both	0.6562 ± 0.0057	0.6739 ± 0.0038	0.6879 ± 0.0052	0.6825 ± 0.0074	0.6746 ± 0.0056

We run ablation experiments to study the combined effects of patch size vs. model with and without node and patch PE, see Table 13 and Table 14.

Overall, increasing the number of patches improves the results independently of using or not the PEs for ZINC and Peptide-func. Node PE clearly helps more than patch PE for both datasets and using both PEs is generally more helpful for a larger number of patches.

F. Study on Different Designs of Graph-Based MHA

Table 15. Different designs of graph-based multi-head attention (gMHA) in transformer layer.

gMHA	Equation	ZINC (MAE↓)	Peptides-func (AP↑)
Standard/Full attention (Vaswani et al., 2017)	$\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$	0.1784 ± 0.0238	0.6778 ± 0.0039
Graph Attention (Dwivedi & Bresson, 2021)	$\text{softmax}\left(A^P \odot \frac{QK^T}{\sqrt{d}}\right)V$	0.1527 ± 0.0067	0.6795 ± 0.0070
Kernel Attention (Mialon et al., 2021)	$\text{softmax}\left(\text{RW}(A^P) \odot \frac{QK^T}{\sqrt{d}}\right)V$	0.1010 ± 0.0031	0.6844 ± 0.0102
Additive Attention (Ying et al., 2021)	$\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V + \text{LL}(A^P)$	0.1632 ± 0.0063	0.6842 ± 0.0057
Hadamard Attention	$\left(A^P \odot \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)\right)V$	0.0849 ± 0.0047	0.6919 ± 0.0085

We conducted experiments on ZINC and Peptides-func datasets to explore five different versions of Graph ViT. The versions primarily differ in the attention function used. The attention functions we considered are as follows: (1) Standard/Full Attention: This attention function is based on the original attention mechanism introduced by Vaswani et al. (2017). (2) Graph Attention: This attention function is derived from the Graph Transformer (GT) model proposed by Dwivedi & Bresson (2021). (3) Kernel Attention: This attention function is based on the kernel attention mechanism proposed by Mialon et al. (2021) in the GraphiT model. (4) Additive Attention: This attention function is derived from the Graphormer model proposed by Ying et al. (2021). (5) Hadamard Attention: We employed Hadamard attention as the default attention function in our Graph ViT model. Results are presented in the Table 15.

Experiments clearly demonstrate that the choice of the self-attention function is important. The Hadamard attention provides the best performance for ZINC (0.0849) and for peptides-func (0.6919) among all attention functions.

Table 16. Effect of data augmentation (DA): ✗ means no DA and ✓ uses DA.

Model	DA	ZINC		Peptides-struct	
		MAE ↓	Time (S/Epoch)	MAE ↓	Time (S/Epoch)
GCN-MLP-Mixer	✗	0.2537 ± 0.0139	5.3603	0.2761 ± 0.0041	6.8297
	✓	0.1347 ± 0.0020	5.6728	0.2486 ± 0.0041	9.2561
GatedGCN-MLP-Mixer	✗	0.2121 ± 0.0172	5.3816	0.2776 ± 0.0020	7.8609
	✓	0.1244 ± 0.0053	5.7786	0.2508 ± 0.0007	9.5830
GINE-MLP-Mixer	✗	0.1389 ± 0.0171	5.3905	0.2792 ± 0.0043	7.8849
	✓	0.0733 ± 0.0014	5.6704	0.2494 ± 0.0007	8.8136
GraphTrans-MLP-Mixer	✗	0.1665 ± 0.0145	6.0039	0.2802 ± 0.0030	9.0999
	✓	0.0773 ± 0.0030	6.1616	0.2480 ± 0.0013	9.7730

G. Effect of Data Augmentation

Then proposed data augmentation (DA) corresponds to newly generated graph patches with METIS at each epoch, while no DA means patches are only generated at the initial epoch and then reused during training. Table 16 presents different results. First, it is clear that DA brings an increase in performance. Second, re-generating graph patches only add to a small amount of training time.

It is worth noting that the drop edge technique we use here is different to the standard data augmentation techniques such as DropEdge (Rong et al., 2019), and G-Mixup (Han et al., 2022b), which either add slightly modified copies of existing data or generate synthetic based on existing data. Our approach is different and actually specific to the Graph MLP-Mixer model.

H. Long Range Graph Benchmark

Table 17. Comparison of our best results from Table 2 with the state-of-the-art Models on large real world datasets (Dwivedi et al., 2022).

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 (1.1×)	6,646 (8.0×)
SAN + LapPE (Chen et al., 2022a)	493k	0.6384 ± 0.0121	80.47 (9.4×)	12,493 (12.4×)	0.2683 ± 0.0043	79.41 (8.8×)	12,226 (14.7×)
SAN + RWSE (Chen et al., 2022a)	500k	0.6439 ± 0.0075	68.44 (8.0×)	19,691 (19.5×)	0.2545 ± 0.0012	70.39 (7.8×)	12,111 (14.5×)
GPS (Rampášek et al., 2022)	504k	0.6562 ± 0.0115	11.83 (1.4×)	6,904 (6.8×)	0.2515 ± 0.0012	11.74 (1.3×)	6,878 (8.3×)
GNN-AK+ (Zhao et al., 2021)	631k	0.6480 ± 0.0089	22.52 (2.6×)	7,855 (7.8×)	0.2736 ± 0.0007	22.11 (2.5×)	7,634 (9.2×)
SUN (Frasca et al., 2022)	508k	0.6730 ± 0.0078	376.66 (43.8×)	18,941 (18.8×)	0.2498 ± 0.0008	384.26 (42.7×)	17,215 (20.7×)
GCN-MLP-Mixer	329k	0.6832 ± 0.0061	8.48	716	0.2486 ± 0.0041	8.12	679
GatedGCN-MLP-Mixer	527k	0.6932 ± 0.0017	8.96	969	0.2508 ± 0.0007	8.44	887
GINE-MLP-Mixer	397k	**0.6970 ± 0.0080**	8.59 (1.0×)	1,010 (1.0×)	0.2494 ± 0.0007	8.51	974
GraphTrans-MLP-Mixer	593k	0.6858 ± 0.0062	9.94	975	0.2480 ± 0.0013	9.00	1,048
GCN-ViT	493k	0.6855 ± 0.0049	8.90	628	0.2468 ± 0.0015	8.55	609
GatedGCN-ViT	692k	0.6942 ± 0.0075	9.07	848	**0.2465 ± 0.0015**	9.00 (1.0×)	833 (1.0×)
GINE-ViT	561k	0.6919 ± 0.0085	8.98	920	0.2449 ± 0.0016	8.77	902
GraphTrans-ViT	757k	0.6876 ± 0.0059	9.94	975	0.2455 ± 0.0027	9.58	981

We have provided additional experiments with the recent Long Range Graph Benchmark (LRGB) (Dwivedi et al., 2022) 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 17.

We summarize the main results as follows: 1) Graph MLP-Mixer sets new SOTA performance with the best scores of 0.6970 on Peptides-fun and 0.2449 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 0.056 Average Precision improvement on Peptides-func and an average 0.028 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, epspecially when training with large graphs, such as SAN+LSPE (Chen et al., 2022a) and SUN (Frasca et al., 2022). For example, SUN gives similar performance to Graph MLP-Mixer, 0.6730 on Peptides-func and 0.2498 on Peptides-struct, but requires 44x memory and 19x training time (Table 17).

I. Mitigating Oversquashing in TreeNeighbourMatch

As discussed in section 5.3, each example of TreeNeighbourMatch is a binary tree of depth r. The goal is to predict an alphabetical label for the target node, where the correct answer is the label of the leaf node that has the same degree as the target node. Figure 2 shows that standard MP-GNNs (i.e., GCN, GGCN, GAT and GIN) fail to generalize on the dataset from $r = 4$, whereas our model mitigates over-squashing and generalizes well until $r = 7$.

To better understand these empirical observations, we first note that as shown by Alon & Yahav (2020), MP-GNNs are fundamentally limited in their ability to solve larger TreeNeighbourMatch cases as they 'squash' information about the graph into the target node's embedding, which can hold a limited amount of information in their floating point representation. Next, we consider Graph MLP-Mixer from an expressiveness point of view, and provide a simple construction to illustrate that it avoids this problem by transmitting long-range information directly without oversquashing. Concretely, consider each node as one patch. Then, Graph MLP-Mixer's Patch Encoder extracts each node's degree and alphabetical label, storing them into the resulting Patch Embeddings. The next Token Mixer layer then compares each node's degree to the target node's, and outputs an indicator variable for whether these degrees are equal, which is transmitted to the next layer. Finally, by combining each node's alphabetical label and this indicator variable, the Fully Connected layer can then output the alphabetical label of the node with matching degree to the target node. In summary, we can observe that Graph MLP-Mixer can solve TreeNeighbourMatch instances while only requiring that each node embedding to capture information about that patch, not the entire graph, thus avoiding the inherent limitations of MP-GNNs as discussed in (Alon & Yahav, 2020).

J. 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 $\{G_1, ..., G_P\}$, 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.

K. Limitations

The current limitations of the model are as follows.

1) Arbitrary choice of the number of clusters in Metis. The number of patches needs to be selected and the number is different across different datasets. Besides, selecting the number of patches to be the same for graphs of variable sizes makes the network operate at different levels of graph resolution and may affect the overall performance.

2) Empirical experiments on WL-expressivity. Our results on the expressivity of the Graph MLP-Mixer are empirical. A theoretical analysis of the expressivity of the model on graphs with higher WL degrees would be valuable but such an analysis is non-trivial.

3) Training and pre-training on large-scale datasets of small and large graphs. More experimental results on Mal-Net (Freitas et al., 2020), PascalVOC-SP, COCO-SP (Dwivedi et al., 2022) and PCQM4Mv2 (Hu et al., 2020) to further test the supervised ability of the model. Besides, the pre-training capability of Graph MLP-Mixer on large-scale datasets with small graphs and large graphs was also not studied. We leave these tasks as future work.