GraFix++: a novel graph transformer based on a fixed multi-head structural attention mechanism

GraFix++: a novel graph transformer based on a fixed multi-head structural attention mechanism

Lingfeng Zhang[a], Luca Cosmo[b], Giorgia Minello[c,b], Andrea Torsello[b], Luca Rossi[a,*]

[a] *The Hong Kong Polytechnic University, Hong Kong*
[b] *Ca' Foscari University of Venice, Italy*
[c] *University of Essex, United Kingdom*

Abstract

Graph neural networks (GNNs) have achieved strong performance on graph learning tasks, but their message-passing mechanism makes it difficult to capture long-range structural dependencies and may lead to over-smoothing in deeper architectures. Graph transformers offer a possible remedy, yet existing approaches typically rely on learnable attention or additional structural encodings, which increase the number of trainable parameters and computational cost while not always exploiting graph structure explicitly. Motivated by these limitations, we propose GraFix++, a graph transformer based on a fixed (non-learnable) multi-head structural attention mechanism derived from graph kernels. Multiple attention heads capture a range of structural similarities between substructures in the input graph, while a GNN is employed to improve the node features extraction. The resulting graph transformer showcases an excellent performance on standard graph classification benchmarks, matching or surpassing a wide range of alternative graph-based approaches. Furthermore, our model benefits from a reduced number of learnable parameters and competitive training runtime. In our experiments, we extensively evaluate the impact of various graph kernels, multiple attention heads, and GNN integration, demonstrating their collective contribution to the model's superior performance.

Keywords:

*Corresponding author: luca.rossi@polyu.edu.hk

Graph neural network, Graph transformer, Graph kernel

1. Introduction

Graphs provide a natural representation for a diverse range of entities, from images [1, 2] and social networks [3] to molecular structures [4, 5], as they capture relational information more effectively than unstructured feature sets. While the non-Euclidean nature of graphs offers significant expressive power, it also poses challenges for conventional deep learning approaches, which struggle to directly operate on such data structures. Indeed, while models like Convolutional Neural Networks (CNNs) [6, 7] and transformers [8] have achieved remarkable success in areas such as computer vision and natural language processing, they are designed to process vectorial data, making them less effective for tasks involving graphs, where structure plays a crucial role.

Specialized neural architectures, such as GCN [9], GraphSAGE [10], GIN [11], and more recently interpretable models such as GKNN [12], have therefore emerged as a result of the attempts to develop deep learning models better suited to operate on graphs. These graph neural networks (GNNs) work by iteratively propagating and aggregating node feature information between adjacent nodes, thereby encoding the topological and structural characteristics of graphs into learned representations. Prior to the development of GNNs, graph kernels [13, 14], such as the subgraph kernel [15] and the Weisfeiler-Lehman (WL) kernel [16], were considered the predominant methodology for applying traditional machine learning techniques to graph data. However, GNNs have demonstrated significant empirical advantages, consistently outperforming graph kernel-based approaches across a wide range of tasks. Nevertheless, GNNs are not without their limitations. A critical challenge lies in their inability to effectively model long-range dependencies between nodes, particularly as the depth of the network increases [17]. This issue is further compounded by the phenomenon of feature over-smoothing, wherein node representations tend to converge to indistinguishable states, thereby undermining the ability of the

model to capture nuanced structural information.

Motivated by these challenges, in recent years a number of researchers have proposed to modify the original transformer architecture to deal with graph inputs [18], allowing them to tap into the transformer ability to model long-range dependencies through the attention mechanism [8]. Standard approaches to allow transformers to harness the structural information of graphs include primarily processing the graph features with a GNN before then passing them to a transformer, or vice-versa [18]. Other approaches include introducing structure-aware positional encodings [19] and, more recently, developing graph-specific attention mechanisms [20, 21] which are naturally able to capture structural information. Inspired by these works, in [22] we have shown that it is possible to replace the learnable attention mechanism of (graph) transformers with a fixed, non-learnable one without negatively impacting the model performance while significantly reducing the training time and number of learnable parameters [22]. Indeed, in some cases fixing the attention coefficients has lead to improvements in the classification performance of model [22].

In the present work, we develop GraFix [22] further, introducing a new hybrid architecture that retains the fixed attention mechanism (extended to deal with multiple attention head as opposed to a single on as in [22]) while incorporating a GNN to compute the value matrix from the original node features (see Section 4). The resulting model (GraFix++) is able to exploit multiple well-known graph kernels to measure the similarity between each node. The transformer then leverages this information when attending to graph nodes. We further enhance our model by replacing the standard linear transformation layer in the transformer (used to compute the value matrix) with a GNN layer. This modification enables the model to better leverage node features, thereby improving its ability to encode and utilize both structural and feature information within the graph. Indeed, while the original architecture [22] is able to effectively capture structural information through the use of graph kernels, it struggle to effectively utilize node-level features, resulting in suboptimal performance on datasets where node features are rich and informative. Through

an extensive set of experiments and an in-depth ablation study, we compare our model with alternative GNNs and graph transformers (including GraFix), demonstrating that GraFix++ achieves competitive graph classification accuracy while reducing the number of model parameters and training time.

In summary, our contributions are as follows:

- We enhance the model capabilities by extending GraFix from a single attention mechanism to a multi-head attention mechanism (GraFix++), introducing different graph kernels to extract structural features of the graph. This in turn enables the model to capture the structural patterns in the graph more comprehensively.

- In the processing of node features, we introduce a single-layer GNN to enhance and enrich the node features, enabling the model to handle more effectively datasets with richer and more complex node information.

- Through an extensive set of experiments, we show that GraFix++ significantly outperforms well-known alternative models on graph classification tasks. Additionally, we conduct an in-depth exploration of the impact of different graph kernels on our model, as well as the effectiveness of the multi-head mechanism.

2. Related Work

In machine learning, graph kernels make it possible to apply kernel-based algorithms, such as support vector machines, to graphs. This is achieved by defining appropriate kernel functions that can capture the structural characteristics of graphs when mapping them to an (implicit) embedding space. Well-known graph kernels include: 1) the Graphlet kernel [23], which counts occurrences of small connected subgraphs called graphlets; 2) the Random Walk kernel, which compares the behaviour of random walks [24] on pairs of input graphs; 3) quantum walk kernels [25], which use quantum [26] rather than random walk to compare the graphs; 4) the Subgraph kernel [15], which compares graphs

by enumerating common subgraphs; 5) the WL kernel [16], which refines node labels iteratively based on neighborhood information and measures similarity by evaluating label consistency across iterations; and 6) shortest-path kernel, which is based on the distributions of shortest path lengths between all pairs of nodes, capturing structural similarities by leveraging path-based features [27].

With the advent of deep neural networks, researchers have increasingly focused on end-to-end models to generalize existing architectures for graph-structured data. GNNs can be seen as an extension of the convolution operation, a key component of CNNs, beyond the regular structure of pixel grids [12]. Most existing GNNs adopt the message-passing paradigm, where nodes exchange information via edges to iteratively update their feature representations. Notable GNN models include GCN [9], GraphSAGE [10], DiffPool [28], DGCNN [29], and GIN [11], with more recent efforts focusing on developing interpretable GNN architectures [12, 30, 31, 32]. As discussed in the previous section, a key limitation of these models is their strong ability to capture local graph structures, often at the expense of effectively modeling long-range node dependencies.

Graph transformers address the inability of standard transformers to take structural information into account by incorporating some level structural awareness. This integration can be primarily achieved through three approaches: 1) passing graph features enriched by GNNs to a transformer, 2) employing unique positional encodings to capture graph structures, and 3) designing more sophisticated attention mechanisms tailored to graph data. For instance, Graph-Trans [33] utilizes a GCN to extract local structural information, which is subsequently fed into a transformer model. Similarly, GraphiT [20] replaces the GCN with a Graph Convolutional Kernel Network (GCKN) [34], leveraging kernel-based methods to encode local graph substructures. In terms of positional encoding, Dwivedi and Bresson [19] propose using the eigenvectors of the graph Laplacian to represent node positions. Building on this, SAN [35] extends the approach by incorporating the entire Laplacian spectrum and learning interactions between different frequency components, thereby enhancing the model's ability to capture complex graph structures. Additionally, Rampášek et

al. [36] introduce a random walk-based positional encoding, further expanding the toolkit for graph representation learning.

A number of graph transformer methods propose to directly modify the attention mechanism to account for structural information. The attention mechanism of the vanilla transformer can be regarded as propagating information over a fully connected graph, thus disregarding the underlying structure of the data. This however can be modified to take the graph structure into account. Dwivedi and Bresson [19] propose to use the adjacency matrix to restrict the attention mechanism to local information about the nodes. Inspired by Tsai et al. [37], who show how the original attention mechanism can be interpreted as the application of a kernel smoother over the input features, GraphiT [20] applies this idea to graphs and uses kernels between graph nodes to modulate the attention mechanism. Chen et al. [21] take this one step further and propose to replace the entire attention mechanism with a kernel smoother, where the attention between pair of nodes is a function of the similarity between the subgraphs rooted at those nodes. In particular, they propose to use an exponential kernel between the features extracted by a GNN on the local subgraphs around each node. Other approaches have sought to bias the attention using the the shortest path distance [38] and the resistance distance [39] between the nodes.

Along these lines, Difformer [40, 41] formalizes the attention mechanism as a constrained diffusion process, showing that standard transformers can be viewed as a specific type of neural diffusion on a complete graph. To improve efficiency and scalability, SGFormer [42] introduces a simplified, one-layer transformer structure that utilizes a simple global attention mechanism to capture long-range dependencies while relying on a GNN for local message passing. Other approaches have sought to bias the attention using the shortest path distance and the resistance distance between the nodes. Gradformer [43] introduces an exponential decay mask into the self-attention mechanism, allowing the model to focus on local details while maintaining the capacity to capture long-range dependencies by correlating attention scores with node proximities. For a comprehensive survey of graph transformers, see [44].

Our model is closely related to [19, 21], as we also formulate the attention mechanism between nodes in terms of graph kernels. However, our attention is fixed, *i.e.*, it is not learned but precomputed as the graph kernel between subgraphs rooted on different nodes. While [19, 21] use GNNs to compute the embeddings of the nodes neighbourhoods, our framework allows to directly use a wide variety of well-known (non-differentiable) graph kernels. Our work is also inspired by [45, 46], which show that fixing the attention coefficients can have little to no negative impact on the model performance (which can actually improve [46]), while it results in a more computationally efficient model [47, 48] thanks to the reduced number of parameters. At the same time, we also compare the impact of different graph kernels on the model and combine multiple graph kernels into a multi-head attention mechanism to improve the model's perception of different structural features. To the best of our knowledge, our work is the first to apply non-learnable attention patterns to graph transformers.

3. Background

Let $G = (V, E, \mathbf{X})$ denote a node-attributed graph over n nodes with edge set E, where $\mathbf{X} \in \mathbb{R}^{n \times d}$ is the matrix of d-dimensional features of the nodes $v \in V$. In this section we introduce the two main ingredients underpinning our model: 1) graph kernels and 2) the transformer multi-head attention mechanism.

3.1. Graph Kernels

Kernel methods are a class of algorithms that allow to learn from pairwise inputs representing the similarity between data points, rather than the individual points themselves. Consider a set X and a positive semi-definite kernel function $k : X \times X \to \mathbb{R}$ such that $k(x, y) = \phi(x)^\top \phi(y)$ for all $x, y \in X$, where $\phi : X \to H$ is a map from X to the Hilbert space H.

While in general X can represent any type of data on which we can define a kernel, in the context of this paper we are interested in kernels operating on a finite set of graphs. A large number of graphs kernels have been introduced

in the literature, the vast majority of which falls under the category of R-convolution kernels [49]. The idea underpinning these kernels is to define the similarity between two graphs in terms of simpler substructures, such as nodes, edges, or subgraphs. These kernels can be either implicit (only the kernel $\mathcal{K}$ is computed) or explicit (the map ϕ is also computed), with explicit kernels being the most common type, *i.e.*,

$$k(G_1, G_2) = \phi(G_1)^\top \phi(G_2)\,, \tag{1}$$

where $\phi(G)$ typically enumerates a certain type of substructures (*e.g.*, nodes, cycles, subtrees) present in G, and the type of these substructures differentiates the various graph kernels. For example, in the WL kernel [16] hash tables obtained by iteratively refining the initial node labels (or their degree, in the case of unattributed graphs) are used to create the graph embedding vectors $\phi(G_1)$ and $\phi(G_2)$.

3.2. The Transformer Architecture

Transformers [8] rose to prominence due to their unmatched performance in language modelling tasks. At its core the transformer architecture revolves around the attention mechanism, which allows the model to learn contextualized representations of the input tokens encapsulating long-range dependencies within the input sequence. Beyond language modelling, transformers have been applied with success to fields including computer vision [50] and speech processing [51], and now form the basis of nearly every modern deep learning architecture.

Given a matrix of input features $\mathbf{X}$, the transformer maps this to the matrices $\mathbf{Q}$ (Query), $\mathbf{K}$ (Key), and $\mathbf{V}$ (Value) through three distinct linear transformations. With these matrices to hand, the attention is then defined as

$$\text{Attention}(\mathbf{X}) = \text{softmax}\Big(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\Big)\mathbf{V}, \tag{2}$$

where $\sqrt{d_k}$ is a scaling factor that depends on the dimension of $\mathbf{Q}$. The updated features $\mathbf{X}'$ are then computed as

$$\mathbf{X}' = \text{Attention}(\mathbf{X}) + \mathbf{X}\,, \tag{3}$$

which can eventually be used to solve the downstream task at hand, *e.g.*, classification. In most modern transformer architectures it has now become common practice to let $\mathbf{Q} = \mathbf{K}$ and to use multiple attention *heads* whereby multiple sets of matrices $\mathbf{Q}$ and $\mathbf{V}$ are learned to allow the network to model multiple contexts. Note also that in a vanilla implementation of the transformer model for graphs, where the input features are the node features, the model inevitably ends up ignoring the crucial topological information intrinsic to the graph structure.

3.3. Transformers as Kernel Machines

Note that in Eq. 2, the dot product $\mathbf{Q}\mathbf{K}^T$ captures the pairwise similarity information between the input tokens. In light of this, Tsai et al. [37] reformulate Eq. 2 through the lens of kernels, rewriting the attention mechanism applied to the i-th input feature x_i as

$$\text{Attention}(x_i) = \sum_{x_j \in \mathbf{X}} \frac{k(x_i, x_j)}{\sum_{x_l} k(x_i, x_l)} f(x_j)\,, \tag{4}$$

where f denotes the value function (in the case of the original transformer, a linear function), k is a kernel function between input features, and we omitted the filtering function of the formulation in [37] for simplicity.

In the case of a graph $G = (V, E, \mathbf{X})$, given a kernel function $k(x_v, x_u)$, where $x_v, x_u \in \mathbf{X}$ are the node features of $v, u \in V$, this yields

$$\text{Attention}(x_v) = \sum_{u \in V} \frac{k(x_v, x_u)}{\sum_{w \in V} k(x_v, x_w)} f(x_u)\,. \tag{5}$$

In Chen et al. [21], the kernel in Eq. 5 is defined by first using a GNN to create enriched features encapsulating the local neighbourhoods of the nodes and then using an exponential kernel function k on these enriched features (see Fig. 1). Dwivedi and Bresson [19] follow a similar but slightly more convoluted approach where two kernels, one capturing the similarity between the features and one capturing the similarity between the nodes positions with a graph, are multiplied together.

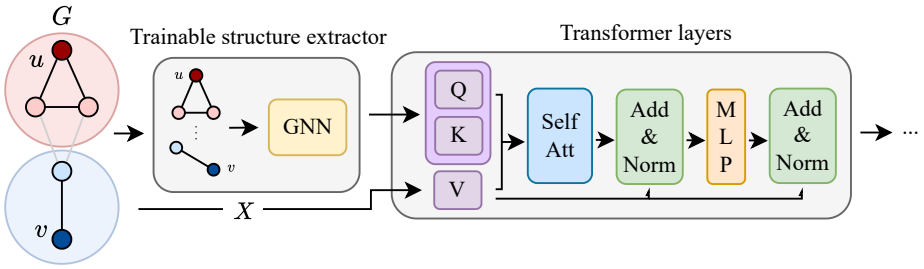


Figure 1: In [21] the authors use a GNN to create enriched features encapsulating the local neighbourhoods and define the keys and queries of the transformer attention mechanism. Note the similarities and difference with our model, GraFix++ (Fig. 2), where the attention is defined by a precomputed set of graph kernels.

Equation (5) can be extended to a multi-head setup by using different kernel functions k_h for each head. Let H denote the number of each heads, and let k_h be the kernel function for the h-th head. The multi-head attention mechanism can be defined as:

$$\text{Attention}_h(x_v) = \sum_{u \in V} \frac{k_h(x_v, x_u)}{\sum_{w \in V} k_h(x_v, x_w)} f_h(x_u), \tag{6}$$

The outputs of all heads are combined to produce the final representation

$$\text{MA}(x_v) = \text{Concat}(\text{Attention}_1(x_v), \ldots, \text{Attention}_H(x_v))W_O\,, \tag{7}$$

where W_O is a learnable weight matrix used to project the concatenated outputs back to the desired dimensionality.

As explained in the next section, we differ from both these approaches in that, rather than learning $k(x_v, x_u)$, we tap into the well-established graph kernel literature, using multiple precomputed graph kernels to measure node similarity. In other words, we propose to fix the attention coefficients rather than learning them end-to-end. Our experiments demonstrate that this leads to a significant reduction in training time (especially when just a single attention head is used) while only minimally affecting the performance of the model.

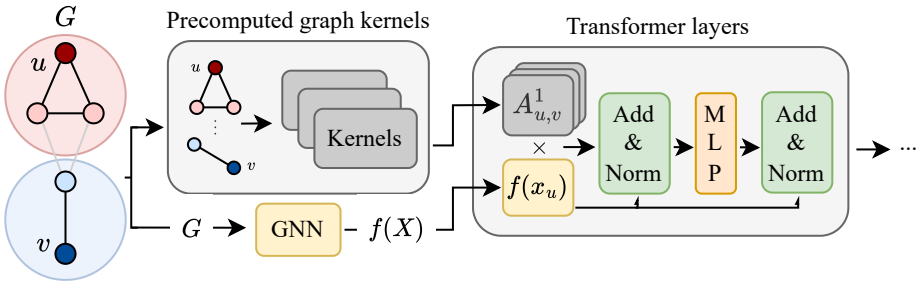


Figure 2: The structure of the proposed graph transformer. Given an input graph G and two of its nodes u and v, the multi-head attention mechanism is defined in terms of the graph kernel similarity between subgraphs rooted around u and v, respectively. In each transformer layer, the attention coefficient $A^h_{u,v}$ between u and v is defined in terms of the h-th kernel output and multiplied by the transformed feature $f(x_u)$ output by a GNN (Eq. 11). The results for all attention heads are then concatenated and projected down to a set dimension (Eq. 7). Finally, the features output by the transformer layer are pooled and passed to a classification head to solve the downstream task.

4. Our Model

Figure 2 illustrates the architecture of GraFix++. The two main components of the proposed model are: 1) the GNN embedding layer and 2) the graph kernel multi-head attention mechanism. This design addresses two limitations of previous approaches, particularly GraFix. The GNN embedding layer introduces a disentanglement between (continuous) node feature representations updated via message passing and the kernel-based structural embeddings, while the multi-head fixed-attention mechanism allows the model to simultaneously capture multiple complementary structural contexts via different graph kernels.

GNN embedding layer. In the vanilla transformer a set of linear transformations is employed to learn the matrices of values, keys, and queries. While GraFix bypasses the need to learn keys and queries by employing a fixed kernel-based attention mechanism [22], the values are still computed as a simple linear transformation of the input node features. This in turn simply increases the dimensionality of the node features without effectively extracting meaningful

information from them. Therefore, we propose replacing the linear transformation layer with a GNN, which allows us to enrich the node features thanks to message-passing mechanism of GNNs.

Let u be a node of the input graph $G = (V, E, \mathbf{X})$. During the k-th message-passing iteration, the node features from the neighborhood $\mathcal{N}(u)$ (*i.e.*, the set of nodes that share an edge with u) is aggregated to yield

$$m_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}^{(k)} \left(\left\{ h_v^{(k)}, \forall v \in \mathcal{N}(u) \right\} \right) , \tag{8}$$

which is then used to update the node feature $h_u^{(k)}$ to

$$h_u^{(k+1)} = \text{UPDATE}^{(k)} \left(h_u^{(k)}, m_{\mathcal{N}(u)}^{(k)} \right) , \tag{9}$$

where UPDATE and AGGREGATE denote learnable differentiable functions and $h_u^{(0)}$ is initialized to $x_u \in \mathbf{X}$.

Multi-head attention. Similarly to GraFix [22], we employed a set of pre-computed graph kernels to define the attention coefficients of our graph transformer. However here we go beyond [22] by extending the original architecture to incorporate multiple attention heads. This in turns enables GraFix++ to go beyond the extraction of a single graph structural pattern, as different graph kernels can effectively capture diverse types of graph structural information, including subgraph structures, path-based features, and neighborhood subgraph characteristics. By leveraging multi-head fixed attention, these varied structural features can be seamlessly integrated, allowing the model to harness a richer and more comprehensive representation of the graph. To this end, we commence by extracting the k-hop subgraph $S_{k,G}(v)$ around each node v, *i.e.*, the subgraph induced by selecting the nodes that can be reached from the central node by traversing at most k edges. Given a set of graph kernel $\mathcal{K} = \{k_1, k_2, \cdots, k_h, \cdots, k_H\}$, we compute the corresponding kernel matrices encapsulating the similarity between all pairs of subgraphs using Eq. 1, where all node features have been one-hot encoded. We stress that this step is pre-computed, *i.e.*, given a training set of graph we extract the k-hop subgraphs

and compute the kernel matrices *prior* to training the network.

With the kernel matrices to hand, we then define the attention coefficients of the graph transformer as follows. For each $k_h \in \mathcal{K}$, the element $A^h_{v,u}$ corresponding to the attention coefficient between the node v and u is

$$A^h_{v,u} = k_h(S_{k,G}(v), S_{k,G}(u)) \,. \tag{10}$$

The attention mechanism used to update the node representations in the transformer layer(s) is then

$$\text{Attention}_h(x_v) = \sum_{u \in V} A^h_{v,u} f(x_u) \,, \tag{11}$$

where $f(x_u)$ denotes the input feature for the node u. For the first transformer layer, this will be equal to the enriched GNN feature for u, while for successive layers k it will be equal to the output of the previous $k-1$ layer.

Note that the attention mechanism of Eq. 11 differs from that Eq. 4 in that we omit the normalization factor which ensures the row-wise stochasticity of the attention matrix. We experimentally observe that this leads to a better performance of the model while not impacting its convergence. Finally the results for different graph kernels are combined together according to Eq. 7, enabling the updated node features to integrate the structural characteristics captured by the kernels. Finally, the node features are passed through an an average graph pooling layer and a multi-layer perceptron to solve the downstream task (*e.g.*, graph classification task). Algorithm 1 shows the pseudocode underpinning the proposed multi-head structural attention mechanism.

4.1. Computational Complexity

Different graph kernels have different computational complexities and, in our model, graph kernels are all precomputed. We take the WL kernel as an example and calculate its time complexity. The computational complexity of the WL kernel can be expressed as $O(slm + s^2ln)$, where s is the number of subgraphs, l represents the iteration count, n corresponds to the maximum number of nodes and m denotes the maximum number of edges within the

Algorithm 1 The proposed multi-head structural attention mechanism.

Input: Graph $G = (V, E, \mathbf{X})$, graph kernels $\mathcal{K} = \{k_1, \ldots, k_H\}$

Extract the k-hop subgraph $S_{k,G}(v)$ for each $v \in V$
for each $k_h \in \mathcal{K}$ **do**
    Compute $A^h_{v,u} = k_h(S_{k,G}(v), S_{k,G}(u))$ for all $u, v \in V$
end for
for each $k_h \in \mathcal{K}$ **do**
    Compute $\text{Attention}_h(x_v) = \sum_{u \in V} A^h_{v,u} f(x_u)$ for all $v \in V$
end for
Compute $\text{MA}(x_v) = \text{Concat}(\text{Attention}_1(x_v), \ldots, \text{Attention}_H(x_v))W_O$
return $\text{MA}(x_v)$ for all $v \in V$

subgraphs. We consider the additional operations introduced by the GNN. The time complexity of a single-layer GNN is $O(|E|d + |V|d^2)$, where $|E|$ is the number of edges, $|V|$ is the number of nodes, and d is the feature dimension. Note that the computation of the WL kernel can be made more efficient by exploiting GPU hardware. We use the `scatter` function of PyTorch Geometric to efficiently aggregate the neighboring labels and then construct histograms to enumerate the labels for each graph, before comparing their similarity.

5. Experimental Results

We compare GraFix++ against GraFix [22] as well as six state-of-the-art GNNs and five different graph transformer methods on graph classification tasks: GCN [9], DGCNN [29], DiffPool [28], ECC [52], GIN [11], GraphSAGE [10], vanilla transformer (node features only) [8], Structure-Aware Transformer (SAT) [21], GraphiT with diffusion coding, GraphiT with adjacency coding, GraphiT with GCKN coding [20]. In addition to [22], GraphiT [20] and SAT [21] are the two most closely related architectures to ours, as they both employ some sort of kernel to replace the standard attention mechanism, although in both cases the attention is not fixed but learned. To further evaluate

the performance on node classification, we also compare our model with additional strong baselines, including GraphTransformer [19], Graphormer [38], GraphGPS [36], NodeFormer [53], Difformer [40, 41] and SGFormer [42]. In our experiments we use four RTX4090 GPUs, each with a memory capacity of 24GB. All our code is written in PyTorch and PyTorch Geometric[1].

5.1. Experimental Datasets

We evaluate our model on six commonly used datasets for graph classification tasks. Five datasets are biological/chemical datasets (MUTAG, PTC, PROTEINS, NCI1, AIDS) [4] and one is a social dataset (IMDB-BINARY) [54]. We also evaluate our model on node classification tasks. For these tasks, we consider three standard citation network datasets: Cora, CiteSeer, and PubMed [55].

Specifically, MUTAG is a collection of mutagenic aromatic and heteroaromatic nitro compounds, commonly used as a benchmark in graph classification tasks to evaluate the ability of models to predict their mutagenic activity based on molecular structures. PTC is a dataset consisting of chemical compounds classified by their carcinogenicity in rodents. In this paper, we specifically utilize the PTC_MR subset, which focuses on male rat data. PROTEINS is a widely used benchmark in graph classification, comprising protein structures represented as graphs, where nodes correspond to amino acids and edges represent interactions, with the task of classifying proteins into enzyme and non-enzyme categories. NCI1 comprises chemical compounds tested for activity against non-small cell lung cancer, with each molecule represented as a graph where nodes denote atoms and edges represent chemical bonds. AIDS consists of chemical compounds screened for activity against HIV, with each molecule represented as a graph with atoms as nodes and covalent bonds as edges. IMDB-BINARY is a dataset of ego networks derived from movie collaborations, where nodes represent actors/actresses and edges indicate co-appearance in the same movie, with the task of classifying the networks into types.

[1] `https://github.com/lcukyfuture/GraFix-`

Table 1: Summary statistics for the six graph datasets used in these experiments.

Dataset	Size	Classes	Avg. num. of nodes	Avg. num. of edges	Has node features
MUTAG	188	2	17.93	19.79	Yes
PTC	344	2	14.29	14.69	Yes
PROTEINS	1,113	2	39.06	72.82	Yes
NCI1	4,110	2	29.87	32.30	Yes
AIDS	2,000	2	15.69	16.20	Yes
IMDB-BINARY	1,000	2	19.77	96.53	No
Cora	1	7	2,708	5,429	Yes
CiterSeer	1	6	3,327	4,723	Yes
PubMed	1	3	19,717	44,338	Yes

Cora is a citation network where nodes represent scientific papers and edges represent citations between them. Each paper is described by a binary word vector indicating the presence of dictionary words, and the task is to classify papers into one of seven research areas. CiteSeer is a similar citation network consisting of scientific publications from the CiteSeer digital library. Nodes are classified into six categories based on their research fields, using sparse bag-of-words features derived from the document text. PubMed comprises scientific publications from the PubMed database pertaining to diabetes. The network consists of nodes categorized into three classes, with node features represented by TF-IDF weighted word frequencies from the abstracts.

We apply one-hot encoding to the node features of all the datasets except for IMDB-BINARY, which does not have node features and thus we use the degree of each node instead. Table 1 shows some of the graph dataset basic statistics, including number of graphs, classes, and average number of nodes and edges.

5.2. Experimental Setup

For the graph classification tasks, we evaluate the performance of each model using 10-fold cross-validation, with a 90/10 training+validation/test split, further splitting train+validation into training (90) and validation (10). For the node classification tasks, we follow the evaluation protocol and experimental

pipeline proposed in OpenGT [44]. In our experimental setup, GraFix++ is constructed by integrating a four-head attention mechanism with GraphConv [56], where the attention mechanism is computed using four widely adopted graph kernels: Random Walk (RW) [24], Shortest Path (SP) [27], Graphlet Sampling (GL) [23], and Weisfeiler-Lehman (WL) [16]—each capturing distinct structural features due to their significant differences in feature extraction. GraphConv, a straightforward yet effective message-passing GNN module, is employed to facilitate information propagation across the graph structure.

We use the training/validation sets to select the optimal model configurations through grid search. To establish the hyperparameter search space, we follow the ranges specified in [57] for GNNs, while for the vanilla transformer and GraphiT we refer to the ranges outlined in [20]. For SAT [21], we vary: 1) the number of GCN layers from 1 to 3; 2) the number of transformer layers from 1 to 3; 3) the number of attention heads in [1, 4, 8]; and 4) the weight decay in [0.01, 0.001, 0.0001]. For our model, we vary: 1) the number of transformer layers from 1 to 3; 2) the number of hops k to build the subgraphs from 1 to 3, with the exception of IMDB-BINARY where the range is limited to 1; 3) the number of WL kernel iterations ranges from 1 to 5, except for the IMDB-BINARY dataset where it is limited to 1 and 2. For the RW kernel we use a step of 2 on all datasets, while for the GL kernel we set the sampling number to 5 on all datasets except for IMDB-BINARY and AIDS, where this is set to 3. See GraKeL [58] for the explanation of these parameters. Finally, as the graph kernels we employ are designed to work with low-dimensional discrete node features, we discretize the node features of the Cora, Cornell, and Pubmed graphs using k-means. We then define the search space for the number of clusters as the range $[16, 256]$, at intervals of 2^n.

For the node classification task baselines, we report the results directly from [44]. For our model, and for all the graph classification baseline, given the grid of hyperparameters, we select the model configuration with the lowest loss on the validation set and we measure the accuracy on the test set. We repeat this procedure for every fold and finally we report the average accuracy $\pm$

standard error over the 10 folds, except for the node classification tasks where we follow [44] and compute the average accuracy ± standard deviation over 3 runs with 3 random seeds. During the training phase, stochastic gradient descent and Adam are employed as optimization algorithms, with initial learning rates set to 0.1 and 0.001, respectively. The learning rate is systematically reduced to half of its current value every 50 epochs.

We set the maximum number of epochs for all models to 1000, with an early stopping mechanism to minimize overfitting. To this end we use a predetermined patience hyperparameter, which was set in alignment with the lowest validation loss recorded within the initial 200 epochs. More specifically, if the model validation loss failed to improve after the 200 epoch mark, this particular loss value was identified as the optimal validation loss.

5.3. Graph Classification Results

Table 2 shows that our model outperforms the alternative ones, both traditional GNNs and transformer-based architectures, on five out of six datasets (MUTAG, PTC, PROTEINS, AIDS, and IMDB-BINARY), while achieving a higher mean accuracy than the vanilla transformer on all datasets.

On the one hand, we observe that GraFix exhibits varying performance across datasets, with its effectiveness closely tied to the nature of the data. Specifically, GraFix tends to underperform on datasets characterized by complex node features, such as NCI1, where the intricate feature representations may not be fully captured by its structural-focused approach. In contrast, GraFix excels on datasets like IMDB-BINARY, where no node features are available. This highlights the model strength in extracting and leveraging structural information, making it suitable for tasks where graph topology plays a dominant role.

On the other hand, GraFix++ demonstrates significant improvements in classification accuracy over the original GraFix framework, particularly on the MUTAG, PTC, and NCI1 datasets. On MUTAG, GraFix++ achieves an accuracy of 87.25 ± 1.5, outperforming all competing methods, including the original GraFix (85.09 ± 2.6). Similarly, on PTC, GraFix++ achieves an accuracy of

Table 2: Average mean classification accuracy ± standard error for our model (GraFix) and the competing ones on the 6 datasets considered in this study. The best model for each dataset is highlighted in bold, while the second best model is underlined.

Dataset	MUTAG	PTC	PROTEINS	NCI1	AIDS	IMDB-BINARY
GCN	75.99 ± 3.5	56.11 ± 2.8	71.60 ± 0.8	74.53 ± 0.7	94.55 ± 0.4	72.80 ± 1.8
GraphConv	82.51 ± 2.9	53.76 ± 2.7	71.37 ± 1.1	76.06 ± 0.5	94.15 ± 0.5	72.90 ± 1.2
DGCNN	85.03 ± 2.7	50.60 ± 2.2	73.86 ± 0.7	74.38 ± 0.7	99.40 ± 0.1	71.90 ± 1.4
DiffPool	81.35 ± 2.5	57.83 ± 2.3	71.25 ± 1.5	**79.32 ± 0.9**	99.15 ± 0.1	70.50 ± 1.2
ECC	80.82 ± 4.2	50.87 ± 2.8	71.17 ± 1.0	75.82 ± 0.5	96.05 ± 0.2	70.40 ± 1.8
GIN	80.26 ± 3.1	58.69 ± 1.6	74.76 ± 0.9	78.89 ± 0.6	99.35 ± 0.2	72.30 ± 1.2
GraphSAGE	72.22 ± 2.8	55.78 ± 2.7	71.25 ± 1.5	77.44 ± 0.7	97.40 ± 0.3	71.30 ± 2.3
Vanilla transformer	72.78 ± 3.6	51.74 ± 2.0	70.98 ± 1.1	65.67 ± 0.7	94.70 ± 0.5	72.40 ± 1.5
SAT	79.68 ± 3.4	56.72 ± 2.2	73.31 ± 1.10	76.27 ± 0.5	96.90 ± 0.4	72.90 ± 1.4
GraphiT + Diffusion	81.46 ± 2.6	60.14 ± 2.1	74.21 ± 1.2	76.69 ± 0.7	96.75 ± 0.5	70.60 ± 1.5
GraphiT + Adj	79.27 ± 1.7	57.84 ± 2.5	71.07 ± 1.4	77.96 ± 0.7	95.85 ± 0.4	71.90 ± 1.4
GraphiT + GCKN	81.93 ± 2.2	59.91 ± 3.2	70.98 ± 1.3	78.91 ± 0.7	98.40 ± 2.5	71.10 ± 1.5
GraFix	85.09 ± 2.6	57.57 ± 2.4	75.11 ± 1.2	71.70 ± 0.5	99.45 ± 0.1	73.30 ± 1.4
GraFix++	**87.25 ± 1.5**	**61.92 ± 1.8**	**76.19 ± 0.8**	76.47 ± 0.6	**99.55 ± 0.1**	**73.40 ± 1.5**

61.92 ± 1.8, surpassing both GraFix (57.57 ± 2.4) and the second-best method, GraphiT + Diffusion (60.14 ± 2.1). While GraFix++ does not achieve the highest accuracy on NCI1, it shows a substantial improvement over GraFix, increasing the accuracy from 71.70 ± 0.5 to 76.47 ± 0.6. This improvement underscores the effectiveness of enriching node features through a single-layer GraphConv, which enhances the model's ability to capture both structural and feature-based information.

5.4. Node Classification Results

The experimental results for node classification on the Cora, CiteSeer, and PubMed datasets are shown in Table 3, where GraFix++ (with either WL, or SP, or both WL & SP kernels) is compared against several representative baselines, including GCN and advanced graph transformers. Our model achieves state-of-the-art performance on the CiteSeer and PubMed datasets with an accuracy of 0.6750 ± 0.0050 and 0.7793 ± 0.0045, respectively, outperforming all competitors including GraphTransformer and SGFormer. As discussed more in detail in subsection 5.8, we observe that the combination of multiple kernels

Table 3: Node classification accuracy ± standard deviation on Cora, CiteSeer, and PubMed datasets. For each dataset, the best (second best) model is highlighted in bold (underlined).

Dataset	Cora	CiteSeer	PubMed
GCN	0.7180 ± 0.0298	0.6173 ± 0.0177	0.7150 ± 0.0297
Graphormer	0.4323 ± 0.0455	0.4360 ± 0.0140	OOM
GraphTransformer	**0.7913 ± 0.0052**	0.6717 ± 0.0062	0.7420 ± 0.0099
GraphGPS	0.7150 ± 0.0051	0.6227 ± 0.0246	0.7327 ± 0.0184
NodeFormer	0.7103 ± 0.0175	0.5903 ± 0.0204	0.6953 ± 0.0144
Difformer	0.7650 ± 0.0099	0.6447 ± 0.0217	0.7587 ± 0.0066
SGFormer	0.7703 ± 0.0021	0.6580 ± 0.0008	0.7347 ± 0.0034
GraFix++ (WL)	0.7507 ± 0.0118	**0.6750 ± 0.0050**	<u>0.7707 ± 0.0040</u>
GraFix++ (SP)	0.7560 ± 0.0060	0.6580 ± 0.0060	0.7450 ± 0.0070
GraFix++ (WL+SP)	<u>0.7720 ± 0.0050</u>	<u>0.6740 ± 0.0080</u>	**0.7793 ± 0.0045**

allows GraFix++ to match or surpass the classification accuracy of the best performing single kernel instance of our model, effectively removing the need to select a specific kernel. Overall, these results demonstrate that the strong performance of our model persists when considering node instead of graph classification tasks.

5.5. Interpretability Analysis

To shed light on the inner workings of our model, we then conduct a structural sensitivity analysis and visualization on the MUTAG dataset using 2-hop subgraphs. In the bar chart of Fig. 3 (right), we show the effect on the loss of masking each head in turn from GraFix++. We can see that the GL kernel emerges as the most influential head, followed by the WL kernel. This in turn is highly consistent with the nature of the MUTAG dataset, where specific molecular sub-structures (specific molecular rings) are definitive indicators of mutagenicity. The removal of GL information causes the most significant performance degradation, proving its critical role in the model inference. While the standalone accuracy of the WL kernel is higher than that of the GL kernel, our ablation study shows that masking the GL kernel results in a more significant

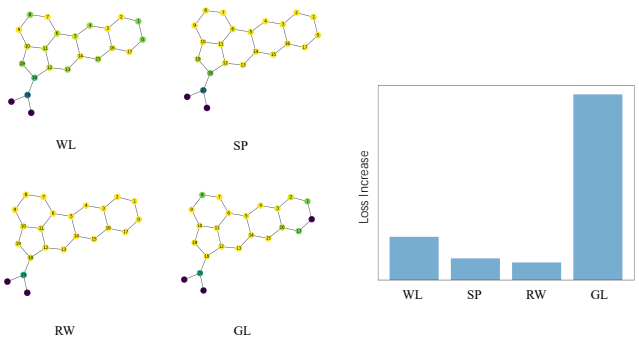


Figure 3: Distribution of the node attention values corresponding to the 4 attention heads (left) and bar chart of the loss increase resulting in masking each of the heads (right).

loss increase. This suggests that while WL is a robust general-purpose descriptor, its information may partially overlap with other kernels like SP or RW. In contrast, the GL kernel captures highly specific, non-redundant structural motifs (specific molecular rings) that the model becomes heavily dependent on in a multi-head setting. Consequently, the GL kernel provides more *unique* discriminative power that other heads cannot easily compensate for, leading to a sharper drop in performance when it is removed.

The graphs in Fig. 3 (left) show instead the value of $\text{Attention}_h(x_v)$ (see Eq. 11) for each node and attention head, with low values indicated in yellow and high values indicated in blue. Once again, the graphs show that the GL kernel captures more unique and discriminative information than the other kernels. The WL head, on the other hand, captures localized atomic environments, providing complementary information to the global-local graphlet patterns.

These interpretability findings align well with our quantitative experimental results, where our model using the WL and GL kernels achieves the highest classification accuracy. This suggests that our multi-head attention mechanism effectively prioritizes the most relevant structural kernels for the given task.

5.6. Runtime Analysis

Table 4 shows that, in terms of training time, GraFix is significantly faster than all the alternative graph transformers, including the vanilla transformer

Table 4: Average per epoch training time ± standard error (measured in seconds $\times 10^{-}2$). For each dataset, the fastest (second fastest) model is highlighted in bold (underlined).

Dataset	MUTAG	PTC	PROTEINS	NCI1	AIDS	IMDB-BINARY
Vanilla transformer	3.76±0.25	<u>5.22±0.26</u>	19.25±0.19	<u>57.77±0.27</u>	**27.94±0.23**	<u>15.76±0.28</u>
SAT	23.64±0.27	43.35±0.28	139.33±0.31	508.17±2.35	127.82±0.30	294.15±0.39
GraphiT + Diffusion	4.44±0.17	6.31±0.27	31.49±0.20	70.82±0.23	34.46±0.23	20.70±0.33
GraphiT + Adj	4.61±0.18	6.26±0.27	31.17±0.22	70.93±0.21	34.19±0.21	19.38±0.30
GraphiT + GCKN	3.89±0.19	6.70±0.22	35.55±0.24	77.36±0.46	35.01±2.5	22.16±0.25
GraFix	**1.89±0.04**	**4.25±0.05**	**11.99±0.04**	**52.63±0.05**	<u>28.20±0.13</u>	**9.97±0.04**
GraFix++	<u>3.16±0.09</u>	5.58±0.10	<u>18.63±0.12</u>	63.15 ± 0.18	30.36±0.21	19.97±0.12

computed on the node features. Indeed, this is achieved by precomputing the attention coefficients rather than learning them end-to-end. The slight increase in running speed observed for GraFix++, on the other hand, can be attributed to the additional computational operations required by the model, resulting from the introduction of multiple attention heads.

At the same time, note that GraFix++ remains significantly faster than both SAT and GraphiT, with the exception of the IMDB-BINARY dataset. Notably, the runtime of GraFix++ is nearly comparable to that of the vanilla transformer. This shows that GraFix++ maintains a substantial advantage in terms of training efficiency while simultaneously enhancing the accuracy of the original GraFix model. Note that the times shown here do not include the precomputation of the graph kernels or the GCKN features, diffusion kernel, and adjacency matrix positional encoding of GraphiT [20]. Finally, these training times refer to the best performing models selected during the grid search and used to generate the classification results of Table 2.

5.7. Number of Model Parameters

We present a comparative analysis of the parameter counts of our proposed model against several alternative graph transformer architectures evaluated in these experiments, as showed in Table 5. As expected, by leveraging a precomputed non-learnable attention matrix, our model achieves a significant reduction in the number of trainable parameters, even when compared to the standard

Table 5: Comparison of the number of model parameters for different configurations of GraphiT and our model. For each dataset, the model with the least (second least) number of parameters is highlighted in bold (underlined).

Dataset	MUTAG	PTC	PROTEINS	NCI1	AIDS	IMDB-BINARY
Vanilla transformer	105,154	105,858	104,898	107,074	107,138	113,410
SAT	142,594	143,298	142,338	144,514	144,578	150,850
GraphiT + Diffusion	104,386	105,090	104,130	106,306	106,370	112,642
GraphiT + Adj	104,386	105,090	104,130	106,306	106,370	112,642
GraphiT + GCKN	111,362	112,066	111,106	113,282	113,346	119,618
GraFix	**67,522**	**68,226**	**67,266**	**69,442**	**69,506**	**75,778**
GraFix++	80,706	82,114	80,194	84,546	84,674	97,218

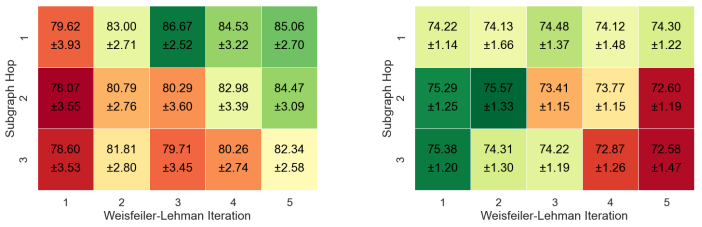
transformer architecture. Importantly, while fixing the attention coefficients reduces the number of trainable parameters within the model, it still maintains or improves the model accuracy. Therefore our model is able to optimize computational efficiency without sacrificing performance. This is particularly relevant when we compare our results to those of SAT, which effectively *learns* a kernel (a dot product attention between learned structural features), which in turn involves complex GNN aggregations during every epoch.

Specifically, we observe that GraFix utilizes fewer than 50% and approximately 65% of the parameters required by SAT and GraphiT, respectively. GraFix++, on the other hand, exhibits an increase in parameter count when compared to to GraFix, however it remains more parameter-efficient than all other transformer-based models and significantly outperforms GraFix in terms of classification accuracy as shown in Table 2. Finally note that for a fair comparison of parameter counts across all models, we standardized the number of transformer layers to 3 in each architecture.

5.8. Ablation Studies

We conclude the experiments by evaluating the impact of different parameters and architectural choices on the performance of our model.

We start by evaluating the impact of varying the number of WL kernel iterations and subgraph hop numbers on the performance of the model (with a



(a) MUTAG (b) PROTEINS

Figure 4: Average classification accuracy ± standard error on the (a) MUTAG and (b) PROTEINS datasets as we vary the subgraph hops k and the number of WL iterations h.

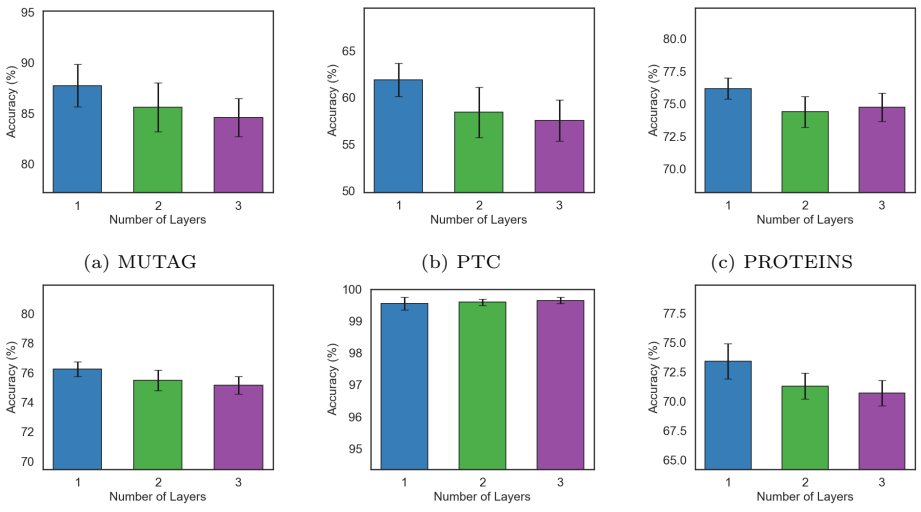
standard linear embedding instead of the GNN one). As illustrated in Figure 4, we report the average classification accuracy of our model across different configurations, varying the number of subgraph hops k from 1 to 3 and the number of WL iterations h from 1 to 5. On both the MUTAG and PROTEINS datasets, our model achieves optimal performance with a relatively small number of WL iterations. This phenomenon can be attributed to the smoothing effect observed in the attention matrix as the number of iterations increases, which consequently reduces the discriminative capacity of structural features. Regarding the subgraph hop parameter k, we observed distinct behaviours between the MUTAG and PROTEINS datasets. Specifically, on the MUTAG dataset, the model performance improves when smaller subgraphs (fewer hops) are utilized, whereas on the PROTEINS dataset, the model demonstrates a preference for larger subgraphs (more hops). This divergence aligns with the inherent characteristics of the datasets, as MUTAG consists of smaller graphs while PROTEINS comprises larger graphs, as detailed in Table 1.

We then evaluate the impact of type of graph kernel used to calculate the attention coefficients, including the Weisfeiler-Lehman (WL), shortest path (SP), random walk (RW), and graphlet (GL) kernels, as well as their combination (WL+SP+RW+GL), without resorting to the initial GNN embedding. We compare this to the model using a single WL attention head plus the initial GNN embedding layer, as well as the full model combining the four fixed attention heads plus the initial GNN embedding layer. Table 6 shows the results

Table 6: Impact of the multiple attention heads and the GNN embedding. The results are shown as average mean classification accuracy ± standard error for each dataset.

Dataset	MUTAG	PTC	PROTEINS	NCI1	AIDS	IMDB-BINARY
GraFix (WL)	85.09 ± 2.6	57.57 ± 2.4	75.11 ± 1.2	71.70 ± 0.5	99.45 ± 0.1	73.30 ± 1.4
GraFix (SP)	81.90 ± 3.2	57.03 ± 2.2	73.89 ± 1.1	72.75 ± 0.9	99.40 ± 0.1	72.40 ± 1.3
GraFix (RW)	75.99 ± 3.3	57.23 ± 2.6	75.39 ± 1.3	67.95 ± 0.1	99.45 ± 0.1	72.90 ± 1.5
GraFix (GL)	84.62 ± 3.4	57.26 ± 1.4	74.57 ± 1.2	70.76 ± 0.6	99.50 ± 0.2	70.50 ± 1.1
GraFix (WL+SP+RW+GL)	85.61 ± 1.7	60.55 ± 3.9	75.47 ± 1.0	72.87 ± 0.6	**99.63 ± 0.2**	72.70 ± 1.6
GraFix (WL) + GraphConv	86.64 ± 2.2	61.30 ± 2.5	75.20 ± 1.5	75.38 ± 0.9	99.30 ± 0.2	73.20 ± 1.3
GraFix++	**87.25 ± 1.5**	**61.92 ± 1.8**	**76.19 ± 0.8**	**76.47** ± 0.6	99.55 ± 0.1	**73.40 ± 1.5**

of this experiment We observe that the performance of GraFix (without GNN embedding layer) tends to reflect the performance of the underlying kernel, particularly in the case of RW kernel, which is known to perform poorly on MUTAG and NCI1 [59]. This in turn can be due to (a) the tottering effect of the RW kernel (which arises when random walks are allowed to move back and forth between the same two nodes, creating artificial high similarity) and (b) the fact that longer walks are heavily downweighted in the RW kernel, thus causing a partial loss of global structural information (unlike other kernels). MUTAG in particular is a small dataset composed of small graphs, which in turn can further exacerbate the issues of the RW kernel. We also note that the the performance differences when using a single kernel are often not statistically significant, which suggests that our architecture can level the performance between the different kernel choices to some extent, *e.g.*, through the use of the downstream classifier head. Most importantly, when combining multiple kernels our model is able to close the gap between the individual kernels, achieving a classification accuracy that matches the best one achieved by using a single kernel, thus effectively removing the need to choose a particular kernel. Indeed, the main performance boost for GraFix++ seems to be the addition of the GNN embedding layer, as the results of GraFix (WL) + GraphConv show. With the exception of AIDS and IMDB-BINARY, where all models performs similarly well, on all the other datasets the addition of the GNN embedding layer is sufficient to bump the performance of GraFix with WL attention head to the second best one. Overall, it



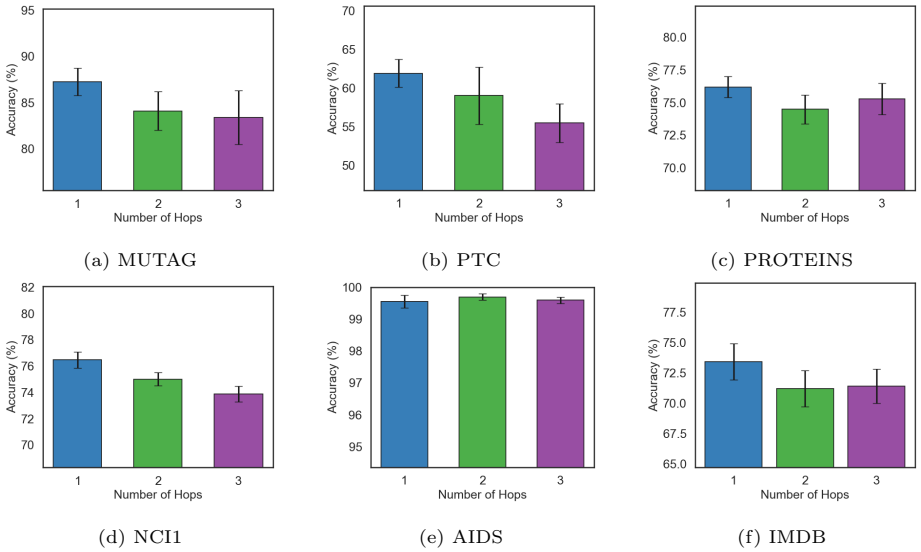
(a) MUTAG (b) PTC (c) PROTEINS

(d) NCI1 (e) AIDS (f) IMDB

Figure 5: Avg classification accuracy ± std error on all datasets for varying number of layers.

is both the combination of the GNN embedding layer and the multiple attention heads that gives GraFix++ an edge over other configurations.

Figure 5 presents an ablation study on the effect of the number of layers on classification accuracy across six datasets with GraFix++. The results indicate that, with the exception of the AIDS dataset, a single-layer model generally achieves higher accuracy. This suggests that by refining the attention mechanism in the transformer and reducing the model training parameters, we enable effective performance with fewer network layers. As a result, compared to other models, GraFix++ achieves superior experimental results with fewer network layers, thereby significantly reducing hardware resource consumption while maintaining a competitive performance. In general, we observe that the optimal number of layers of our GraFix++ remains mostly independent of the nature of the underlying graphs. Indeed, the same behaviour persists on a wide range of datasets from different domains and varied sizes and connectivity structures. A notable exception is the case of node classification datasets. Here the graphs are significantly larger, have a distinctive scale-free nature, and are characterized by high-dimensional node features that are highly discriminative for



(a) MUTAG (b) PTC (c) PROTEINS

(d) NCI1 (e) AIDS (f) IMDB

Figure 6: Avg classification accuracy ± std error on all datasets for varying number of hops.

classifying the nodes. On these datasets, the optimal number of layers selected during the hyperparameters grid search is 2.

Finally, we analyze the impact of varying the number of hops in GraFix++. Figure 6 presents the classification accuracy as the number of hops used to build the subgraphs increases from 1 to 3. For datasets with a relatively small number of nodes, such as MUTAG and PTC, we observe that increasing the number of hops leads to a decline in classification accuracy. This can be attributed to the fact that extracting multi-hop subgraphs in small graphs results in kernel values becoming overly smooth and similar, thereby reducing the discriminative power of the model. This effect is also clearly visible on IMDB-BINARY, where the graphs are relatively small world (*i.e.*, their diameter is small) and thus a few hops are sufficient for a subgraph to span the entire graph. As a consequence, using a single hop allows for better feature extraction while avoiding excessive information blending. We observe a slightly different trend on the AIDS and PROTEINS datasets, where leveraging more hops enables the model to capture a broader range of relational dependencies without excessive smoothing. As a result, the performance remains relatively stable across different hop settings.

6. Conclusion

In this paper, we introduced GraFix++, a novel graph transformer architecture that diverges from conventional approaches by employing a fixed attention mechanisms derived from graph kernels, rather than learning attention weights. Specifically, the attention coefficients are precomputed to capture the similarity between substructures within the input graphs, leveraging the expressive power of graph kernels. We demonstrate that the proposed model, which utilizes multiple graph kernels to compute node-level attention coefficients, achieves competitive performance compared to both GNNs and traditional graph transformers on standard graph classification benchmarks. The modular design of GraFix++ allows for the seamless integration of alternative graph kernels to adapt to diverse datasets, as well as the incorporation of additional fixed attention heads to enrich the extraction of substructure information. Furthermore, our architecture significantly reduces the number of trainable parameters, thereby lowering computational complexity and alleviating hardware resource requirements, while maintaining a robust performance.

While GraFix++ offers substantial improvements in training efficiency and scalability, it still has limitations. Our approach involves a strategic trade-off: the significant reduction in per-epoch training time is achieved at the expense of an initial pre-computation overhead for the structural kernels. Consequently, GraFix++ is optimized for scenarios prioritizing high-throughput training rather than total pipeline latency minimization (including pre-processing).

7. Acknowledgements

This work was funded by the Ministry of University and Research (MUR) under Notice no. 1233 of 01 August 2023 - Italian Fund for Applied Sciences (FISA) – "Low Individual Value Entities Processing in Agricultural Sorting and Selection" - Code FISA-2023-00158 – CUP of project H73C25000330006 – pursuant to Executive Decree no. 6948 of 04/15/2025 of the MUR. The project was also supported by the PRIN 2022 project N. 2022AL45R2 (EYE-FI.AI,

CUP H53D2300350-0001). G.M. acknowledges financial support from project iNEST (Interconnected NordEst Innovation Ecosystem), funded by European Union Next - GenerationEU - National Recovery and Resilience Plan (NRRP) - MISSION 4 COMPONENT 2, INVESTMENT N. ECS00000043 - CUP N. H43C22000540006. In this regard, the manuscript reflects only the authors' views and opinions, neither the European Union nor the European Commission can be considered responsible for them. G.M. also acknowledges financial support from the DoE 2023-2027 (MUR, AIS.DIP.ECCELLENZA2023_27.FF).

References

[1] H. Senior, G. Slabaugh, S. Yuan, L. Rossi, Graph neural networks in vision-language image understanding: A survey, The Visual Computer (2024) 1–26.

[2] A. Gallagher-Syed, H. Senior, O. Alwazzan, E. Pontarini, M. Bombardieri, C. Pitzalis, M. J. Lewis, M. R. Barnes, L. Rossi, G. Slabaugh, Biox-cpath: Biologically-driven explainable diagnostics for multistain ihc computational pathology, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2025.

[3] Y. Leng, L. Yu, Incorporating global and local social networks for group recommendations, Pattern Recognition 127 (2022) 108601.

[4] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, M. Neumann, Tudataset: A collection of benchmark datasets for learning with graphs, in: ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020), 2020.

[5] J. Yu, T. Xu, Y. Rong, J. Huang, R. He, Structure-aware conditional variational auto-encoder for constrained molecule optimization, Pattern Recognition 126 (2022) 108581.

[6] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern recognition 77 (2018) 354–377.

[7] J. Guo, K. Han, H. Wu, Y. Tang, X. Chen, Y. Wang, C. Xu, Cmt: Convolutional neural networks meet vision transformers, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022, pp. 12175–12185.

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

[9] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, in: Proceedings of the 5th International Conference on Learning Representations (ICLR), 2017.

[10] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, Advances in neural information processing systems 30 (2017).

[11] K. Xu, W. Hu, J. Leskovec, S. Jegelka, How powerful are graph neural networks?, in: International Conference on Learning Representations, 2019.

[12] L. Cosmo, G. Minello, A. Bicciato, M. M. Bronstein, E. Rodolà, L. Rossi, A. Torsello, Graph kernel neural networks, IEEE Transactions on Neural Networks and Learning Systems (2024).

[13] N. M. Kriege, F. D. Johansson, C. Morris, A survey on graph kernels, Applied Network Science 5 (2020) 1–42.

[14] Y. Liu, L. Rossi, A. Torsello, A novel graph kernel based on the wasserstein distance and spectral signatures, in: S+SSPR, Springer, 2022, pp. 122–131.

[15] N. M. Kriege, P. Mutzel, Subgraph matching kernels for attributed graphs, in: Proceedings of the 29th International Conference on Machine Learning (ICML), 2012, pp. 315–322.

[16] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, K. M. Borgwardt, Weisfeiler-lehman graph kernels., Journal of Machine Learning Research 12 (9) (2011).

[17] Q. Li, Z. Han, X.-M. Wu, Deeper insights into graph convolutional networks for semi-supervised learning, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 32, 2018.

[18] A. Shehzad, F. Xia, S. Abid, C. Peng, S. Yu, D. Zhang, K. Verspoor, Graph transformers: A survey, IEEE Transactions on Neural Networks and Learning Systems (2026).

[19] V. P. Dwivedi, X. Bresson, A generalization of transformer networks to graphs, arXiv preprint arXiv:2012.09699 (2020).

[20] G. Mialon, D. Chen, M. Selosse, J. Mairal, Graphit: Encoding graph structure in transformers, arXiv preprint arXiv:2106.05667 (2021).

[21] D. Chen, L. O'Bray, K. Borgwardt, Structure-aware transformer for graph representation learning, in: International Conference on Machine Learning, PMLR, 2022, pp. 3469–3489.

[22] L. Zhang, L. Cosmo, G. Minello, A. Torsello, L. Rossi, Grafix: A graph transformer with fixed attention based on the wl kernel, in: International Conference on Pattern Recognition, Springer, 2024, pp. 435–450.

[23] N. Shervashidze, S. Vishwanathan, T. Petri, K. Mehlhorn, K. Borgwardt, Efficient graphlet kernels for large graph comparison, in: Artificial intelligence and statistics, PMLR, 2009, pp. 488–495.

[24] M. Sugiyama, K. Borgwardt, Halting in random walk kernels, Advances in neural information processing systems 28 (2015).

[25] L. Bai, L. Rossi, A. Torsello, E. R. Hancock, A quantum jensen–shannon graph kernel for unattributed graphs, Pattern Recognition 48 (2) (2015) 344–355.

[26] L. Bai, L. Cui, M. Li, P. Ren, Y. Wang, L. Zhang, P. S. Yu, E. R. Hancock, Aegk: Aligned entropic graph kernels through continuous-time quantum walks, IEEE Transactions on Knowledge and Data Engineering 37 (3) (2025) 1064–1078.

[27] K. M. Borgwardt, H.-P. Kriegel, Shortest-path kernels on graphs, in: Fifth IEEE international conference on data mining (ICDM'05), IEEE, 2005, pp. 8–pp.

[28] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, J. Leskovec, Hierarchical graph representation learning with differentiable pooling, Advances in neural information processing systems 31 (2018).

[29] M. Zhang, Z. Cui, M. Neumann, Y. Chen, An end-to-end deep learning architecture for graph classification, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 32, 2018.

[30] A. Feng, C. You, S. Wang, L. Tassiulas, Kergnns: Interpretable graph neural networks with graph kernels, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 36, 2022, pp. 6614–6622.

[31] A. Bicciato, L. Cosmo, G. Minello, L. Rossi, A. Torsello, Classifying me softly: A novel graph neural network based on features soft-alignment, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2022, pp. 43–53.

[32] A. Bicciato, L. Cosmo, G. Minello, L. Rossi, A. Torsello, Gnn-lofi: a novel graph neural network through localized feature-based histogram intersection, Pattern Recognition 148 (2024) 110210.

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

[34] D. Chen, L. Jacob, J. Mairal, Convolutional kernel networks for graph-structured data, in: International Conference on Machine Learning, PMLR, 2020, pp. 1576–1586.

[35] D. Kreuzer, D. Beaini, W. Hamilton, V. Létourneau, P. Tossou, Rethinking graph transformers with spectral attention, Advances in Neural Information Processing Systems 34 (2021) 21618–21629.

[36] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, D. Beaini, Recipe for a general, powerful, scalable graph transformer, Advances in Neural Information Processing Systems 35 (2022) 14501–14515.

[37] Y.-H. H. Tsai, S. Bai, M. Yamada, L.-P. Morency, R. Salakhutdinov, Transformer dissection: An unified understanding for transformer's attention via the lens of kernel, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 4344–4353.

[38] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, T.-Y. Liu, Do transformers really perform badly for graph representation?, Advances in neural information processing systems 34 (2021) 28877–28888.

[39] B. Zhang, S. Luo, L. Wang, D. He, Rethinking the expressive power of gnns via graph biconnectivity, in: International Conference on Learning Representations (ICLR), 2023.

[40] Q. Wu, C. Yang, W. Zhao, Y. He, D. Wipf, J. Yan, Difformer: Scalable (graph) transformers induced by energy constrained diffusion, in: International Conference on Learning Representations (ICLR), 2023.

[41] Q. Wu, D. Wipf, J. Yan, Transformers from diffusion: A unified framework for neural message passing, Journal of Machine Learning Research (JMLR) (2025).

[42] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, J. Yan, Sgformer: Simplifying and empowering transformers for large-graph representations, Advances in Neural Information Processing Systems 36 (2023) 64753–64773.

[43] C. Liu, Z. Yao, Y. Zhan, X. Ma, S. Pan, W. Hu, Gradformer: graph transformer with exponential decay, in: International Joint Conference on Artificial Intelligence, 2024, pp. 2171–2179.

[44] J. Tang, Z. Wang, S. Chen, S. Zhou, J. Chen, J. Bu, Opengt: A comprehensive benchmark for graph transformers, arXiv preprint arXiv:2506.04765 (2025).

[45] R. Child, S. Gray, A. Radford, I. Sutskever, Generating long sequences with sparse transformers, arXiv preprint arXiv:1904.10509 (2019).

[46] A. Raganato, Y. Scherrer, J. Tiedemann, Fixed encoder self-attention patterns in transformer-based machine translation, Vol. EMNLP 2020 of Findings of ACL, Association for Computational Linguistics, 2020, pp. 556–568.

[47] C. P. Chen, Z. Liu, Broad learning system: An effective and efficient incremental learning system without the need for deep architecture, IEEE transactions on neural networks and learning systems 29 (1) (2017) 10–24.

[48] G. Menghani, Efficient deep learning: A survey on making deep learning models smaller, faster, and better, ACM Computing Surveys 55 (12) (2023) 1–37.

[49] D. Haussler, et al., Convolution kernels on discrete structures, Tech. rep., Citeseer (1999).

[50] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16x16 words: Transformers for image recognition at scale, in: International Conference on Learning Representations, 2021.

[51] L. Dong, S. Xu, B. Xu, Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition, in: 2018 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, 2018, pp. 5884–5888.

[52] M. Simonovsky, N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 3693–3702.

[53] Q. Wu, W. Zhao, Z. Li, D. P. Wipf, J. Yan, Nodeformer: A scalable graph structure learning transformer for node classification, Advances in Neural Information Processing Systems 35 (2022) 27387–27401.

[54] P. Yanardag, S. Vishwanathan, Deep graph kernels, in: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, 2015, pp. 1365–1374.

[55] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI magazine 29 (3) (2008) 93–93.

[56] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, M. Grohe, Weisfeiler and leman go neural: Higher-order graph neural networks, in: Proceedings of the AAAI conference on artificial intelligence, Vol. 33, 2019, pp. 4602–4609.

[57] F. Errica, M. Podda, D. Bacciu, A. Micheli, A fair comparison of graph neural networks for graph classification (2020).

[58] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, M. Vazirgiannis, Grakel: A graph kernel library in python, Journal of Machine Learning Research 21 (54) (2020) 1–5.

[59] G. Nikolentzos, G. Siglidis, M. Vazirgiannis, Graph kernels: A survey, Journal of Artificial Intelligence Research 72 (2021) 943–1027.