Global Self-Attention as a Replacement for Graph Convolution

Md Shamim Hussain
hussam4@rpi.edu
Rensselaer Polytechnic Institute
Troy, New York, USA

Mohammed J. Zaki
zaki@cs.rpi.edu
Rensselaer Polytechnic Institute
Troy, New York, USA

Dharmashankar Subramanian
dharmash@us.ibm.com
IBM T. J. Watson Research Center
Yorktown Heights, New York, USA

ABSTRACT

We propose an extension to the transformer neural network architecture for general-purpose graph learning by adding a dedicated pathway for pairwise structural information, called edge channels. The resultant framework – which we call Edge-augmented Graph Transformer (EGT) – can directly accept, process and output structural information of arbitrary form, which is important for effective learning on graph-structured data. Our model exclusively uses global self-attention as an aggregation mechanism rather than static localized convolutional aggregation. This allows for unconstrained long-range dynamic interactions between nodes. Moreover, the edge channels allow the structural information to evolve from layer to layer, and prediction tasks on edges/links can be performed directly from the output embeddings of these channels. We verify the performance of EGT in a wide range of graph-learning experiments on benchmark datasets, in which it outperforms Convolutional/Message-Passing Graph Neural Networks. EGT sets a new state-of-the-art for the quantum-chemical regression task on the OGB-LSC PCQM4Mv2 dataset containing 3.8 million molecular graphs. Our findings indicate that global self-attention based aggregation can serve as a flexible, adaptive and effective replacement of graph convolution for general-purpose graph learning. Therefore, convolutional local neighborhood aggregation is not an essential inductive bias.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Artificial intelligence.*

KEYWORDS

graph neural networks, graph representation learning, self-attention

ACM Reference Format:
Md Shamim Hussain, Mohammed J. Zaki, and Dharmashankar Subramanian. 2022. Global Self-Attention as a Replacement for Graph Convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539296

KDD '22, August 14–18, 2022, Washington, DC, USA.

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539296

1 INTRODUCTION

Graph-structured data are ubiquitous in different areas such as communication networks, molecular structures, citation networks, knowledge bases and social networks. Due to the flexibility of the structural information in graphs, they are powerful tools for compact and intuitive representation of data originating from a very wide range of sources. However, this flexibility comes at the cost of added complexity in processing and learning from graph-structured data, due to the arbitrary nature of the interconnectivity of the nodes. Recently the go-to solution for deep representation learning on graphs has been Graph Neural Networks (GNNs) [17, 34]. The most commonly used GNNs follow a convolutional pattern whereby each node in the graph updates its state based on that of its neighbors [24, 42] in each layer. On the other hand, the pure self-attention based transformer architecture [38] has displaced convolutional neural networks for more regularly arranged data, such as sequential (e.g., text) and grid-like (images) data, to become the new state-of-the-art, especially in large-scale learning. Transformers have become the de-facto standard in the field of natural language processing, where they have achieved great success in a wide range of tasks such as language understanding, machine translation and question answering. The success of transformers has translated to other forms of unstructured data in different domains such as audio [8, 28] and images [7, 13] and also on different (classification/generation, supervised/unsupervised) tasks.

Transformers differ from convolutional neural networks in some important ways. A convolutional layer aggregates a localized window around each position to produce an output for that position. The weights that are applied to the window are independent of the input, and can therefore be termed as *static*. Also, the sliding/moving window directly follows the structure of the input data, i.e., the sequential or grid-like pattern of positions. This is an apriori assumption based on the nature of the data and how it should be processed, directly inspired by the filtering process in signal processing. We call this assumption the *convolutional inductive bias*. On the other hand, in the case of a transformer encoder layer, the internal arrangement of the data does not directly dictate how it is processed. Attention weights are formed based on the queries and the keys formed at each position, which in turn dictate how each position aggregates other positions. The aggregation pattern is thus global and input dependent, i.e., it is *dynamic*. The positional information is treated as an input to the network in the form of positional encodings. In their absence, the transformer encoder is permutation equivariant and treats the input as a multiset. Information is propagated among different positions only via the global self-attention mechanism, which is agnostic to the internal arrangement of the data. Due to this property of global self-attention, distant points in the data can interact with each other as efficiently as nearby points. Also, the network *learns* to form appropriate aggregation

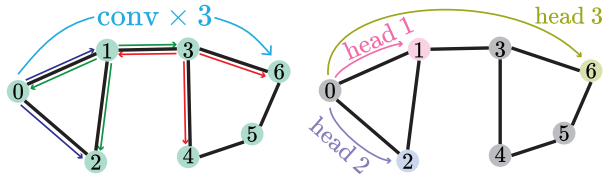

Figure 1: A conceptual demonstration of Graph Convolution (left) and Global Self-Attention (right). It takes three stages of convolution for node 0 to aggregate node 6. With global self-attention, the model can learn to do so in a single step. The attention heads are formed dynamically for a given graph.

patterns during the training process, rather than being constrained to a predetermined pattern.

Although it is often straightforward to adopt the transformer architecture for regularly structured data such as text and images by employing an appropriate positional encoding scheme, the highly arbitrary nature of structure in graphs makes it difficult to represent the position of each node only in terms of positional encodings. Also, it is not clear how edge features can be incorporated in terms of node embeddings. For graph-structured data, the edge/structural information can be just as important as the node information, and thus we should expect the network to process this information hierarchically, just like the node embeddings. To facilitate this, we introduce a new addition to the transformer, namely *residual edge channels* – a pathway that can leverage structural information. This is a simple yet powerful extension to the transformer framework in that it allows the network to directly process graph-structured data. This addition is also very general in the sense that it facilitates the input of structural information of arbitrary form, including edge features, and can handle different variants of graphs such as directed and weighted graphs in a systematic manner. Our framework can exceed the results of widely used Graph Convolutional Networks on datasets of moderate to large sizes, in supervised benchmarking tasks while maintaining a similar number of parameters. But our architecture deviates significantly from convolutional networks in that it does not impose any strong inductive bias such as the convolutional bias, on the feature aggregation process. We rely solely on the global self-attention mechanism to learn how best to use the structural information, rather than constraining it to a fixed pattern. Additionally, the structural information can evolve over layers and the network can potentially form new structures. Any prediction on the structure of the graph, such as link prediction or edge classification, can be done directly from the outputs of edge channels. However, these channels do add to the quadratic computational and memory complexity of global self-attention, with respect to the number of nodes, which restricts us to moderately large graphs. In addition to the edge channels, we generalize GNN concepts like gated aggregation [4], degree scalers [12] and positional encodings [15] for our framework.

Our experimental results indicate that given enough data and with the proposed edge channels, the model can utilize global self-attention to learn the best aggregation pattern for the task at hand. Thus, our results indicate that following a fixed convolutional aggregation pattern whereby each node is limited to aggregating its closest neighbors (based on adjacency, distance, intimacy, etc.) is not an essential inductive bias. With the flexibility of global self-attention, the network can learn to aggregate distant parts of the input graph in just one step as illustrated in Fig. 1. Since this pattern is learned rather than being imposed by design, it increases the expressivity of the model. Also, this aggregation pattern is dynamic and can adapt to each specific input graph. Similar findings have been reported for unstructured data such as images [11, 13, 32]. Some recent works have reported global self-attention as a means for better generalization or performance by improving the expressivity of graph convolutions [31, 40]. Very recently, Graphormer [43] performed well on graph level prediction tasks on molecular graphs by incorporating edges with specialized encodings. However, it does not directly process the edge information and therefore does not generalize well to edge-related prediction tasks. By incorporating the edge-channels, we are the first to propose global self-attention as a direct and general replacement for graph convolution for node-level, link(edge)-level and graph-level prediction, on all types of graphs.

2 RELATED WORK

In relation to our work, we discuss self-attention based GNN models, where the attention mechanism is either constrained to a local neighborhood (local self-attention) of each node or unconstrained over the whole input graph (global self-attention). Methods like Graph Attention Network (GAT) [39] and Graph Transformer (GT) [14] constrain the self-attention mechanism to local neighborhoods of each node only, which is reminiscent of the graph convolution/local message-passing process. Several works have attempted to adopt the global self-attention mechanism for graphs as well. Graph-BERT [45] uses a modified transformer framework on a sampled linkless subgraph (i.e., only node representations are processed) around a target node. Since the nodes do not inherently bear information about their interconnectivity, Graph-BERT uses several types of relative positional encodings to embed the information about the edges within a subgraph. Graph-BERT focuses on unsupervised representation learning by training the model to predict a single masked node in a sampled subgraph. GROVER [33] used a modified transformer architecture with queries, keys and values produced by Message-Passing Networks, which indirectly incorporate the input structural information. This framework was used to perform unsupervised learning on molecular graphs only. Graph Transformer [6] and Graphormer [43] directly adopt the transformer framework for specific tasks. Graph Transformer separately encodes the nodes and the relations between nodes to form a fully connected view of the graph which is incorporated into a transformer encoder-decoder framework for graph-to-sequence learning. Graphormer incorporates the existing structure/edges in the graph as an attention bias, formed from the shortest paths between pairs of nodes. It focuses on graph-level prediction tasks on molecular graphs (e.g., classification/regression on molecular graphs). Unlike these models which handle graph structure in an ad-hoc manner and only for a specific problem, we directly incorporate graph structure into the transformer model via the edge channels and propose a general-purpose learning framework for graphs based only on the global self-attention mechanism, free of the strong inductive bias of convolution. Apart from being used for node feature aggregation,

attention has also been used to form metapaths in heterogeneous graphs, such as the Heterogeneous Graph Transformer (HGT) [22] and the Graph Transformer network (GTN) [44]. However, these works are orthogonal to ours since metapaths are only relevant in the case of heterogeneous graphs and these methods use attention *specifically* to combine heterogeneous edges, over multiple hops. We focus only on homogeneous graphs, but more importantly, we use attention as a global aggregation mechanism.

3 NETWORK ARCHITECTURE

3.1 Preliminaries

The transformer architecture was proposed by Vaswani et al. [38] as a purely attention-based model. The transformer encoder uses self-attention to communicate information between different positions, and thus produces the output embeddings for each position. In the absence of positional encodings, this process is permutation equivariant and treats the input embeddings as a multiset.

Each layer in the transformer encoder consists of two sublayers. The key component of the transformer is the multihead self-attention mechanism which takes place in the first sublayer, which can be expressed as:

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \tilde{\mathbf{A}}\mathbf{V} \tag{1}$$

$$\text{Where,} \quad \tilde{\mathbf{A}} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \tag{2}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are the keys, queries and values formed by learned linear transformations of the embeddings and d_k is the dimensionality of the queries and the keys. $\tilde{\mathbf{A}}$ is known as the (softmax) attention matrix, formed from the scaled dot product of queries and keys. This process is done for multiple sets of queries, keys and values, hence the name multihead self-attention. The second sublayer is the feedforward layer which serves as a pointwise non-linear transformation of the embeddings.

3.2 Edge-augmented Graph Transformer (EGT)

The EGT architecture (Fig. 2) extends the original transformer architecture. The permutation equivariance of the transformer is ideal for processing the node embeddings in a graph because a graph is invariant under the permutation of the nodes, given that the edges are preserved. We call the residual channels present in the original transformer architecture *node channels*. These channels transform a set of input node embeddings $\{h_1^0, h_2^0, ..., h_N^0\}$ into a set of output node embeddings $(h_i^L)_{final}$ (for $1 \leq i \leq N$), where $h_i^\ell \in \mathbb{R}^{d_h}$, d_h is the node embeddings dimensionality, N is the number of nodes, and L is the number of layers. Our contribution to the transformer architecture is the introduction of *edge channels*, which start with an embedding for each *pair of nodes*. Thus, there are $N \times N$ input edge embeddings $e_{11}^0, e_{12}^0, ..., e_{1N}^0, e_{21}^0, ..., e_{NN}^0$ where, $e_{ij}^l \in \mathbb{R}^{d_e}$, d_e is the edge embeddings dimensionality. The input edge embeddings are formed from graph structural matrices and edge features. We define a graph structural matrix as any matrix with dimensionality $N \times N$, which can completely or partially define the structure of a graph (e.g., adjacency matrix, distance matrix). The edge embeddings are updated by EGT in each layer and finally, it produces a set of output edge embeddings $(e_{ij}^L)_{final}$ (for $1 \leq i, j \leq N$) from which

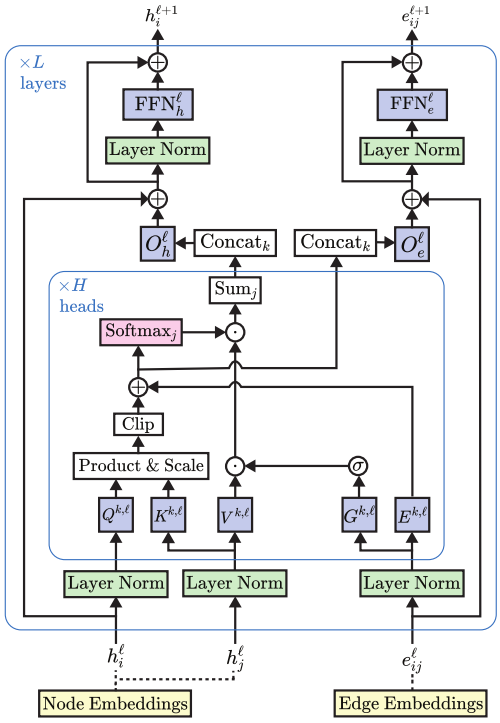

Figure 2: Edge-augmented Graph Transformer (EGT)

structural predictions such as edge labeling and link prediction can be performed.

From equations (1) and (2) we see that the attention matrix is comparable with a row-normalized adjacency matrix of a directed weighted complete graph. It dictates how the node features in a graph are aggregated, similarly to GCN [24]. Unlike the input graph, this graph is dynamically formed by the attention mechanism. However, the basic transformer does not have a direct way to incorporate the input structure (existing edges) while forming these weighted graphs, i.e., the attention matrices. Also, these dynamic graphs are collapsed immediately after the aggregation process is done. To remedy the first problem we let the edge channels participate in the aggregation process as follows (as shown in Fig. 2) – in the ℓ'th layer and for the k'th attention head,

$$\text{Attn}(\mathbf{Q}_h^{k,\ell}, \mathbf{K}_h^{k,\ell}, \mathbf{V}_h^{k,\ell}) = \tilde{\mathbf{A}}^{k,\ell}\mathbf{V}_h^{k,\ell} \tag{3}$$

$$\text{Where,} \quad \tilde{\mathbf{A}}^{k,\ell} = \text{softmax}(\hat{\mathbf{H}}^{k,\ell}) \odot \sigma(\mathbf{G}_e^{k,\ell}) \tag{4}$$

$$\text{Where,} \quad \hat{\mathbf{H}}^{k,\ell} = \text{clip}\left(\frac{\mathbf{Q}_h^{k,\ell}(\mathbf{K}_h^{k,\ell})^T}{\sqrt{d_k}}\right) + \mathbf{E}_e^{k,\ell} \tag{5}$$

where $\odot$ denotes elementwise product. $\mathbf{E}_e^{k,\ell}, \mathbf{G}_e^{k,\ell} \in \mathbb{R}^{N \times N}$ are concatenations of the learned linear transformed edge embeddings. $\mathbf{E}_e^{k,\ell}$ is a bias term added to the scaled dot product between the queries and the keys. It lets the edge channels influence the attention process. $\mathbf{G}_e^{k,\ell}$ drives the sigmoid $\sigma(\cdot)$ function and lets the edge channels also *gate* the values before aggregation, thus controlling the flow of information between nodes. The scaled dot product is clipped to a limited range which leads to better numerical stability

(we used $[-5, +5]$). To ensure that the network takes advantage of the full-connectivity the attention process is randomly masked by adding $-\infty$ to the inputs to the softmax with a small probability during training (i.e., random attention masking). Another approach is to apply dropout [37] to the attention matrix.

To let the structural information evolve from layer to layer, the edge embeddings are updated by a learnable linear transformation of the inputs to the softmax function. The outputs of the attention heads are also mixed by a linear transformation. To facilitate training deep networks, Layer Normalization (LN) [1] and residual connections [19] are used. We adopted the Pre-Norm architecture whereby normalization is done immediately before the weighted sublayers [41] rather than after, because of its better optimization characteristics. So, $\hat{h}_i^\ell = \mathrm{LN}(h_i^{\ell-1})$, $\hat{e}_{ij}^\ell = \mathrm{LN}(e_{ij}^{\ell-1})$. The residual updates can be expressed in an elementwise manner as:

$$\hat{\hat{h}}_i^\ell = h_i^{\ell-1} + \mathbf{O}_h^\ell \Big\|_{k=1}^{H} \sum_{j=1}^{N} \tilde{\mathbf{A}}_{ij}^{k,\ell} (\mathbf{V}^{k,\ell} \hat{h}_i^\ell) \tag{6}$$

$$\hat{\hat{e}}_{ij}^\ell = e_{ij}^{\ell-1} + \mathbf{O}_e^\ell \Big\|_{k=1}^{H} \hat{\mathbf{H}}_{ij}^{k,\ell} \tag{7}$$

Here, $\|$ denotes concatenation. $\mathbf{O}_h^\ell \in \mathbb{R}^{d_h \times d_h}$ and $\mathbf{O}_e^\ell \in \mathbb{R}^{d_e \times H}$ are the learned output projection matrices, with edge embeddings dimensionality d_e and H attention heads.

The feed-forward sublayer following the attention sublayer consists of two consecutive pointwise fully connected linear layers with a non-linearity such as ELU [10] in between. The updated embeddings are $h_i^\ell = \hat{\hat{h}}_i^\ell + \mathrm{FFN}_\mathrm{h}^\ell(\mathrm{LN}(\hat{\hat{h}}_i^\ell))$, $e_i^\ell = \hat{\hat{e}}_i^\ell + \mathrm{FFN}_\mathrm{e}^\ell(\mathrm{LN}(\hat{\hat{e}}_i^\ell))$. The Pre-Norm architecture also ends with a layer normalization over the final embeddings as $(h_i^L)_{final} = \mathrm{LN}(h_i^L)$, $(e_{ij}^L)_{final} = \mathrm{LN}(e_{ij}^L)$.

3.3 Dynamic Centrality Scalers

The attention mechanism in equation (3) is a weighted average of the gated node values, which is agnostic to the degree of the nodes. However, we may want to make the network sensitive to the degree/centrality of the nodes, in order to make it more expressive when distinguishing between non-isomorphic (sub-)graphs, similar to GIN [42]. While this can be achieved by directly encoding the degrees of the nodes as an additional input like [43], we aimed for an approach that is adaptive to the dynamic nature of self-attention. Corso et al. [12] propose scaling the aggregated values by a function of the degree of the node, more specifically a logarithmic degree scaler. But it is tricky to form a notion of degree/centrality for the dynamically formed graph represented by the attention matrix because this row-normalized matrix bears no notion of degree. In our network, the sigmoid gates control the flow of information to a particular node which are derived from the edge embeddings. So we use the sum of the sigmoid gates as a measure of centrality for a node and scale the aggregated values by the logarithm of this sum. With centrality scalers, equation (6) becomes:

$$\hat{\hat{h}}_i^\ell = h_i^{\ell-1} + \mathbf{O}_h^\ell \Big\|_{k=1}^{H} s_i^{k,l} \sum_{j=1}^{N} \tilde{\mathbf{A}}_{ij}^{k,\ell} (\mathbf{V}^{k,\ell} \hat{h}_i^\ell) \tag{8}$$

$$\text{Where,} \quad s_i^{k,l} = \ln\left(1 + \sum_{j=1}^{N} \sigma(\mathbf{G}^{k,\ell} \mathbf{e}_{i,j})\right) \tag{9}$$

Here, $s_i^{k,l}$ is the centrality scaler for node i, for attention head k at layer ℓ. As pointed out by Ying et al. [43], with the addition of a centrality measure the global self-attention mechanism becomes at least as powerful as the 1-Weisfeiler-Lehman (1-WL) isomorphism test and potentially even more so, due to aggregation over multiple hops. Note that commonly used convolutional GNNs like GIN are at most as powerful as the 1-WL isomorphism test [42].

3.4 SVD-based Positional Encodings

While applying the transformer on regularly arranged data such as sequential (e.g., text) and grid-like (e.g., images) data it is customary to use sinusoidal positional encodings introduced by Vaswani et al. [38]. However, the arbitrary nature of structure in graphs makes it difficult to devise a consistent positional encoding scheme. Nonetheless, positional encodings have been used for GNNs to embed global positional information within individual nodes and to distinguish isomorphic nodes and edges [29, 36]. Inspired by matrix factorization based node embedding methods for graphs [3], Dwivedi et al. [15] proposed to use the k smallest non-trivial eigenvectors of the Laplacian matrix of the graph as positional encodings. However, since the Laplacian eigenvectors can be complex-valued for directed graphs, this method is more relevant for undirected graphs which have symmetric Laplacian matrices. To remedy this we propose a method, that is more general and applies to all variants of graphs (e.g., directed, weighted). We propose a form of positional encoding based on precalculated SVD of the graph structural matrices. We use the largest r singular values and corresponding left and right singular vectors to form our positional encodings. We use the adjacency matrix $\mathbf{A}$ (with self-loops) as the graph structural matrix, but it can be generalized to other structural matrices since the SVD of any real matrix produces real singular values and vectors.

$$\mathbf{A} \overset{\text{SVD}}{\approx} \mathbf{U}\Sigma\mathbf{V}^T = (\mathbf{U}\sqrt{\Sigma}) \cdot (\mathbf{V}\sqrt{\Sigma})^T = \hat{\mathbf{U}}\hat{\mathbf{V}}^T \tag{10}$$

$$\hat{\mathbf{\Gamma}} = \hat{\mathbf{U}} \,\|\, \hat{\mathbf{V}} \tag{11}$$

Where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{N \times r}$ matrices contain the r left and right singular vectors as columns, respectively, corresponding to the top r singular values in the diagonal matrix $\Sigma \in \mathbb{R}^{r \times r}$. Here, $\|$ denotes concatenation along the columns. From (10) we see that the dot product between i'th row of $\hat{\mathbf{U}}$ and j'th row of $\hat{\mathbf{V}}$ can approximate $\mathbf{A}_{ij}$ which denotes whether there is an edge between nodes i and j. Thus, the rows of $\hat{\mathbf{\Gamma}}$, namely $\hat{\gamma}_1, \hat{\gamma}_2, ..., \hat{\gamma}_N$, each with dimensionality $\hat{\gamma}_i \in \mathbb{R}^{2r}$, bear denoised information about the edges and can be used as positional encodings. Note that this form of representation based on the dot product is consistent with the scaled dot product attention used in the transformer framework. Since the signs of corresponding pairs of left and right singular vectors can be arbitrarily flipped, we randomly flip the signs of $\hat{\gamma}_i$ during training for better generalization. Instead of directly adding $\hat{\gamma}_i$ to the input embeddings of the node i, we add a projection $\gamma_i = \mathbf{W}_{enc}\hat{\gamma}_i$, where $\mathbf{W}_{enc} \in \mathbb{R}^{d_h \times 2r}$ is a learned projection matrix. This heuristically leads to better results. Since our architecture directly takes structure as input via the edge channels, the inclusion of positional encodings is optional for most tasks. However, positional encodings can help distinguish isomorphic nodes [46] by serving as an absolute global coordinate system. Thus, they make the model more expressive.

However, the absolute coordinates may, in theory, hamper generalization, because they are specific to a particular reference frame that depends on the input graph. But in practice, we did not find any detrimental effect on the performance for any task.

3.5 Embedding and Prediction

Given an input graph, both node and edge feature embeddings are formed by performing learnable linear transformations for continuous vector values, or vector embeddings for categorical/discrete values. In the case of multiple sets of features, their corresponding embeddings are added together. When positional encodings γ_i are used, they are added to the input node embeddings. The edge embeddings are formed by adding together the embeddings from the graph structural matrix and the input edge feature embeddings (when present). For non-existing edges, a masking value/vector is used in the place of an edge feature. As input structural matrix, we use the distance matrix clipped up to k-hop distance, i.e., $\mathbf{D}^{(k)}$ where $\mathbf{D}^{(k)}_{ij} \in \{0, 1, ..., k\}$ are the shortest distances between nodes i and j, clipped to a maximum value of k. We use vector embeddings of the discrete values contained in these matrices.

For node and edge classification/regression tasks, we apply a few final MLP layers on the final node and edge embeddings, respectively, to produce the output. For graph-level classification/regression we adopt one of two different methods. In *global average pooling* method, all the output node embeddings are averaged to form a graph-level embedding, on which final linear layers are applied. In *virtual nodes* method, q new virtual nodes with learnable input embeddings $h^0_{N+1}, h^0_{N+2}, ..., h^0_{N+q}$ are passed through EGT along with existing node embeddings. There are also q different learnable edge embeddings $\tilde{e}_i$ which are used as follows – the edge embedding between a virtual node i and existing graph node j is assigned $e^0_{ij} = e^0_{ji} = \tilde{e}_i$, and the edge embeddings between two virtual nodes i, j, are assigned $e^0_{ij} = e^0_{ji} = \frac{1}{2}(\tilde{e}_i + \tilde{e}_j)$. Finally, the graph embedding is formed by concatenating the output node embeddings of the virtual nodes. This method is more flexible and better suited for larger models. The centrality scalers mentioned above are not applied to the virtual nodes, because by nature these nodes have high levels of centrality which are very different from the graph nodes. So a fixed scaler value of $s^{k,l}_i = 1$ is used instead for these virtual nodes.

For smaller datasets, we found that adding a secondary *distance prediction objective* alongside the graph-level prediction task in a multi-task learning setting serves as an effective means of regularization and thus improves the generalization of the trained model. This self-supervised objective is reminiscent of the unsupervised link prediction objective often used to pre-train GNNs to form node embeddings. In our case, we take advantage of the fact that we have output edge embeddings from the edge channels (alongside the node embeddings, which are used for graph-level prediction). We thus pass the output edge embeddings through a few (we used three) MLP layers and set the distance matrix up to ν-hop, $\mathbf{D}^{(\nu)}$, as a categorical target. Hops greater than ν are ignored while calculating the loss. The loss from this secondary objective is multiplied by a small factor κ and added to the total loss. Note that in this case we always use the adjacency matrix, rather than the distance matrix as the input graph structural matrix so that the edge channels do not simply learn an identity transformation. We emphasize that this objective is only potentially beneficial as a regularization method for smaller datasets by guiding the aggregation process towards a Breadth-First Search pattern, which is a *soft* form of the convolutional bias. In the presence of enough data, the network is able to learn the best aggregation pattern for the given primary objective, which also generalizes to unseen data.

4 EXPERIMENTS AND RESULTS

We evaluate the performance of our proposed EGT architecture in a supervised and inductive setting. We focus on a diverse set of supervised learning tasks, namely, node and edge classification, and graph classification and regression. We also experiment on the transfer learning performance of EGT.

Datasets: In the medium-scale supervised learning setting, we experimented with the benchmarking datasets proposed by Dwivedi et al. [15], namely PATTERN (14K synthetic graphs, 44-188 nodes/graph) and CLUSTER (12K synthetic graphs, 41-190 nodes/graph) for node classification; TSP (12K synthetic graphs, 50-500 nodes/graph) for edge classification; and MNIST (70K superpixel graphs, 40-75 nodes/graph), CIFAR10 (60K superpixel graphs, 85-150 nodes/graph) and ZINC (12K molecular graphs, 9-37 nodes/graph) for graph classification/regression. To evaluate the performance of EGT at large-scale we consider the graph regression task on the PCQM4M and its updated version PCQM4Mv2 datasets [20] which contain 3.8 million molecular graphs with 1-51 nodes/graph. We also experimented on tranfer learning from PCQM4Mv2 dataset to the graph classification tasks on OGB [21] datasets MolPCBA (438K molecular graphs, 1-332 nodes/graph) and MolHIV (41K molecular graphs, 2-222 nodes/graph).

Evaluation Setup: We use the PyTorch [30] numerical library to implement our model. Training was done in a distributed manner on a single node with 8 NVIDIA Tesla V100 GPUs (32GB RAM/GPU), and 2 20-core 2.5GHz Intel Xeon CPUs (768GB RAM). Masked attention was used to process mini-batches containing graphs of different numbers of nodes. This allowed us to use highly parallel tensor operations on the GPU. The results are evaluated in terms of accuracy, F1 score, Mean Absolute Error (MAE), Average Precision (AP), or Area Under the ROC Curve (AUC), as recommended for each dataset. Hyperparameters were tuned on the validation set. Full details of hyperparameters are included in the appendix and the code is available at https://github.com/shamim-hussain/egt.

4.1 Medium-scale Performance

For the benchmarking datasets, we follow the training setting suggested by Dwivedi et al. [15] and evaluate the performance of EGT for a given parameter budget. Comparative results are presented in Table 1. All datasets except PATTERN and CLUSTER include edge features. From the results, we see that EGT outperforms other GNNs (including GAT and GT which use local self-attention, and Graphormer which uses global self-attention but without edge channels) on all datasets except CIFAR10. We see a high level of overfitting for all models on CIFAR10, including our model which overfits the training dataset due to its higher capacity. The edge

Table 1: Experimental results on 6 benchmarking datasets from Dwivedi et al. [15]. Results on PATTERN and CLUSTER datasets are given in terms of weighted accuracy. Red: best model, Violet: good model; arrow next to a metric indicates whether higher or lower is better. Results not shown are not available for that method.

Model	PATTERN % Accuracy ↑		CLUSTER % Accuracy ↑	MNIST % Accuracy ↑	CIFAR10 % Accuracy ↑	TSP F1 ↑		ZINC MAE ↓	
	#Param ≈100K	#Param ≈500K	#Param ≈500K	#Param ≈100K	#Param ≈100K	#Param ≈100K	#Param ≈500K	#Param ≈100K	#Param ≈500K
GCN [24]	63.880 ± 0.074	71.892 ± 0.334	68.498 ± 0.976	90.705 ± 0.218	55.710 ± 0.381	0.630 ± 0.001		0.459 ± 0.006	0.367 ± 0.011
GraphSage [18]	50.516 ± 0.001	50.492 ± 0.001	63.844 ± 0.110	97.312 ± 0.097	65.767 ± 0.308	0.665 ± 0.003		0.468 ± 0.003	0.398 ± 0.002
GIN [42]	85.590 ± 0.011	85.387 ± 0.136	64.716 ± 1.553	96.485 ± 0.097	55.255 ± 1.527	0.656 ± 0.003		0.387 ± 0.015	0.526 ± 0.051
GAT [39]	75.824 ± 1.823	78.271 ± 0.186	70.587 ± 0.447	95.535 ± 0.205	64.223 ± 0.455	0.671 ± 0.002		0.475 ± 0.007	0.384 ± 0.007
GT [14]		84.808 ± 0.068	73.169 ± 0.622						0.226 ± 0.014
GatedGCN [4]	84.480 ± 0.122	86.508 ± 0.085	76.082 ± 0.196	97.340 ± 0.143	67.312 ± 0.311	0.808 ± 0.003	0.838 ± 0.002	0.375 ± 0.003	0.214 ± 0.013
PNA [12]	86.567 ± 0.075			97.690 ± 0.022	70.350 ± 0.630			0.188 ± 0.004	0.142 ± 0.010
DGN [2]	86.680 ± 0.034				**72.700 ± 0.540**			0.168 ± 0.003	
Graphormer [43]		86.650 ± 0.033	74.660 ± 0.236	97.905 ± 0.176	65.978 ± 0.579		0.698 ± 0.007		0.122 ± 0.006
EGT	**86.816 ± 0.027**	**86.821 ± 0.020**	**79.232 ± 0.348**	**98.173 ± 0.087**	68.702 ± 0.409	**0.822 ± 0.000**	**0.853 ± 0.001**	**0.143 ± 0.011**	**0.108 ± 0.009**

Table 2: Results on OGB-LSC PCQM4M and PCQM4Mv2 datasets in terms of Mean Absolute Error (lower is better). Results not shown are not available.

Model	#Param	PCQM4M Validate	PCQM4M Test	PCQM4Mv2 Validate	PCQM4Mv2 Test-dev
GCN [24]	2.0M	0.1684	0.1838	0.1379	0.1398
GIN [42]	3.8M	0.1536	0.1678	0.1195	0.1218
GCN-VN [16, 24]	4.9M	0.1510	0.1579	0.1153	0.1152
GIN-VN [16, 42]	6.7M	0.1396	0.1487	0.1083	0.1084
GINE-VN [5, 16]	13.2M	0.1430			
DeeperGCN-VN [16, 27]	25.5M	0.1398			
GT [14]	0.6M	0.1400			
GT (bigger model) [14]	83.2M	0.1408			
$\text{Graphormer}_{\text{SMALL}}$ [43]	12.5M	0.1264			
Graphormer [43]	47.1M	0.1234	**0.1328**	0.0906	
$\text{EGT}_{\text{Small}}$ (6 layers)	11.5M	0.1260		0.0899	
$\text{EGT}_{\text{Medium}}$ (18 layers)	47.4M	**0.1224**		0.0881	
$\text{EGT}_{\text{Large}}$ (24 layers)	89.3M			**0.0869**	**0.0872**

Table 3: Results on OGB Mol datasets. EGT uses transfer learning from PCQM4Mv2, whereas GIN-VN and Graphormer use transfer learning from PCQM4M. AP stands for Average Precision and AUC for Area Under the ROC Curve, higher is better for both. Results not shown are not available.

Model	MolPCBA #Param	MolPCBA Test AP(%)	MolHIV #Param	MolHIV Test AUC(%)
DeeperGCN-FLAG [25, 27]	6.55M	28.42 ± 0.43	532K	79.42 ± 1.20
DeeperGCN-VN-FLAG [16, 25, 27]	6.55M	28.42 ± 0.43		
PNA [12]	6.55M	28.38 ± 0.35	326K	79.05 ± 1.32
DGN [2]	6.73M	28.85 ± 0.30	110K	79.70 ± 0.97
GINE-VN [5, 16]	6.15M	29.17 ± 0.15		
PHC-GNN [26]	1.69M	29.47 ± 0.26	114K	79.34 ± 1.16
GIN-VN [16, 42] (pre-trained)	3.4M	29.02 ± 0.17	3.3M	77.80 ± 1.82
Graphormer-FLAG [43] (pre-trained)	119.5M	**31.40 ± 0.34**	47.2M	**80.51 ± 0.53**
$\text{EGT}_{\text{Larger}}$ (30 layers) (pre-trained)	110.8M	29.61 ± 0.24	110.8M	**80.60 ± 0.65**

channels allow us to use the distance prediction objective in a multi-task learning setting, which helps lessen the overfitting problem on CIFAR10, ZINC and MNIST. Also, the output embeddings of the edge channels are directly used for edge classification on the TSP dataset which leads to very good results. Note that, Graphormer, which also uses global self-attention but does not have such edge channels, performs satisfactorily for other tasks but not so much on edge classification on the TSP dataset. Since we do not take advantage of the convolutional inductive bias our model shows various levels of overfitting on these medium-sized datasets. While EGT still outperforms other GNNs, we posit that it would further exceed the performance level of convolutional GNNs if more training data were present (we confirm this in the next section). Also, the results indicate that convolutional aggregation is not an essential inductive bias, and global attention can learn to make the best use of the structural information.

4.2 Large-scale Performance

The results for the graph regression task on the OGB-LSC PCQM4M and PCQM4Mv2 datasets [20] are presented in Table 2. We show results for EGT models of small, medium and large network sizes based on number of parameters (details are included in the appendix). Note that the PCQM4M dataset was later deprecated in favor of PCQM4Mv2. So its test labels are no longer available and results are given over the validation set. We include these results for a thorough comparison with established models that report their results on the older dataset. We see that EGT achieves a much lower MAE than *all* the convolutional and local self-attention based (i.e., GT [14]) GNNs. Its performance even exceeds Graphormer [43], which is also a global self-attention based model and can be thought of as an ablated variant of EGT with specialized encodings, such as centrality, spatial and edge encodings and requires similar training time and resources. We hypothesize that EGT gets a better result than Graphormer because of a combination of several factors, including its edge channels, unique gating mechanism and dynamic centrality scalers. Our model is currently the best performing model on the PCQM4Mv2 leaderboard. These results show the scalability of our framework and further confirm that given enough

Table 4: Comparison of results for two ablated variants of EGT (EGT-Constrained and EGT-Simple), along with the complete architecture with (EGT) and without (EGT w/o PE) SVD based positional encodings

	PATTERN % Accuracy ↑	CLUSTER % Accuracy ↑	MNIST % Accuracy ↑	CIFAR10 % Accuracy ↑	TSP F1 ↑	ZINC MAE ↓	PCQM4Mv2 MAE ↓
Model	**#Param≈500K**	**#Param≈500K**	**#Param≈100K**	**#Param≈100K**	**#Param≈500K**	**#Param≈500K**	**#Param≈11.5M**
EGT-Constrained	86.629 ± 0.041	76.701 ± 0.257	96.823 ± 0.204	65.192 ± 0.475	0.846 ± 0.001	0.174 ± 0.004	0.0934
EGT-Simple	86.813 ± 0.013	79.182 ± 0.213	98.148 ± 0.139	64.967 ± 1.263	0.831 ± 0.002	0.228 ± 0.020	0.0900
EGT w/o PE	86.812 ± 0.031	77.665 ± 0.343	**99.218 ± 0.219**	**68.555 ± 0.624**	**0.853 ± 0.001**	0.187 ± 0.005	0.0901
EGT	**86.821 ± 0.020**	**79.232 ± 0.348**	**98.173 ± 0.087**	**68.702 ± 0.409**	**0.853 ± 0.001**	**0.108 ± 0.009**	**0.0899**

data, global self-attention based aggregation can outperform local convolutional aggregation.

4.3 Transfer Learning Performance

In order to experiment on the transferability of the representations learned by EGT, we take an EGT model pre-trained on the large-scale PCQM4Mv2 molecular dataset and fine-tune the weights on the OGB molecular datasets MolPCBA and MolHIV. Although the validation performance improvement seems to plateau for larger models on the PCQM4Mv2 dataset at a certain point, we found that larger pre-trained models perform better when fine-tuned on smaller datasets, so we select the largest model ($\text{EGT}_{\text{Larger}}$) with 30 layers for transfer learning experiments (it achieves a validation MAE of 0.0869 on PCQM4Mv2, same as $\text{EGT}_{\text{Large}}$). The results are presented in Table 3. We see that both EGT and Graphormer achieve comparable results which exceed convolutional GNNs. Graphormer uses pre-trained models from PCQM4M and they found it essential to use the FLAG training method [25] to achieve good fine-tuning results. FLAG uses an inner optimization loop to augment the node embeddings by adding adversarial perturbations to them. However, we do not use any form of specialized training during the fine-tuning process. This is due to two reasons - firstly, we wanted to evaluate our model in the conventional transfer learning setting where the weights of a pre-trained model are simply fine-tuned on a new dataset for a very few epochs which saves training time and resources – whereas, FLAG training takes several times longer training time with additional FLAG hyperparameter tuning. Another reason is that FLAG is an adversarial perturbation method for node embeddings and since we have both node and edge embeddings (including non-existing edges) it is not clear how this method should be adopted for our model – which requires further research.

4.4 Ablation Study

Our architecture is based upon two important ideas – global self-attention based aggregation and residual edge channels. To analyze the importance of these two features, we experiment with two ablated variants of EGT: i) **EGT-Simple:** incorporates global self-attention, but instead of having dedicated residual channels for edges, it directly uses a linear transformation of the input edge embeddings e_{ij}^0 (formed from adjacency matrix and edge features) to guide the self-attention mechanism. The absence of edge channels means that the edge embeddings e_{ij} are not updated from layer to layer. So, edge classification is performed by applying MLP layers on pairwise node-embeddings. It is architecturally similar to Graphormer [43]. While it is slightly less expensive in terms of computation and memory, it still scales quadratically with the number of nodes. ii) **EGT-Constrained** limits the self-attention process to the 1-hop neighborhood of each node, which allows us to compare global self-attention to convolutional local self-attention based aggregation. Also, it only keeps track of the edge embeddings e_{ij} in the edge channels if there is an edge from node i to node j or $i = j$ (self-loop). Architecturally, this variant is similar to GT [14] and can take advantage of the sparsity of the graph to reduce computational and memory costs. More details about these variants can be found in the appendix.

Table 5: Ablation study on the PCQM4Mv2 dataset for $\text{EGT}_{\text{Small}}$ (from Table 2).

Gated Aggregation	Attention Dropout	Virtual Nodes	Centrality Scalers	Positional Encodings	Validate MAE ↓
–	–	–	–	–	0.0965
✓	–	–	–	–	0.0943
✓	✓	–	–	–	0.0926
✓	✓	✓	–	–	0.0919
✓	✓	✓	✓	–	0.0900
✓	✓	✓	✓	✓	**0.0899**

The results for the ablated variants are presented in Table 4. We see that, EGT-Simple can come close to EGT, but is especially subpar when the targeted task is related to edges (e.g., edge classification on the TSP dataset) or when the distance objective cannot be applied (ZINC, CIFAR10) due to the lack of dedicated edge channels. Both EGT-Simple and EGT enjoy an advantage over EGT-Constrained on the large PCQM4Mv2 dataset due to their global aggregation mechanism. This indicates that given enough data, global self-attention based aggregation can outperform local self-attention based aggregation. Additionally to demonstrate the effect of the SVD based positional encodings we include results without positional encodings. Note that the positional encodings lead to a significant improvement for the ZINC and the CLUSTER datasets, but slight/no improvement in other cases. This is consistent with our statement that the positional encodings are optional for our model on some tasks, but their inclusion can often lead to a performance improvement.

To further examine the contribution of different features of our model we carried out a series of experiments on the PCQM4Mv2 dataset for the smallest EGT network. The results are presented in Table 5. We see that the use of gates during aggregation leads to a significant improvement. Another important contributing factor is dropout on the attention matrix which encourages the network to

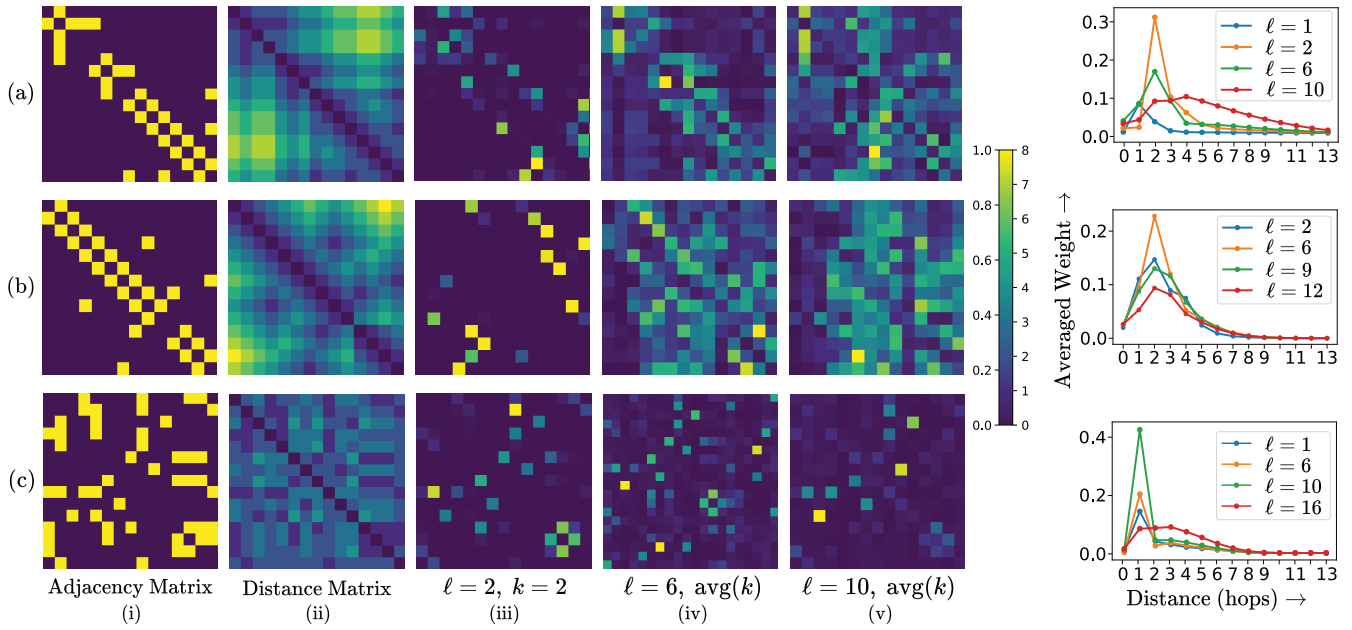

Figure 3: Analysis of aggregation patterns on three datasets – (a) ZINC, (b) PCQM4Mv2, (c) TSP. Left to right – adjacency (i) and distance matrices (ii), an example attention head (iii), average of attention heads in a middle layer (iv) and in a deeper layer (v) – for a particular input graph in the validation set (matrices have been cropped for the TSP dataset). On the right – weights assigned for different hops in different layers, averaged over all heads and all nodes in all the graphs in the validation set.

take advantage of long-distance interactions. The dynamic centrality scalers also help by making the network more expressive. Virtual nodes and positional encodings lead to a more modest performance improvement.

4.5 Analysis of Aggregation Patterns

To understand how global self-attention based aggregation translates to performance gains we examined the attention matrices dynamically formed by the network. These matrices dictate the weighted aggregation of the nodes and thus show how each node is looking at other nodes. This is demonstrated in Fig. 3. We show the adjacency matrix and the distance matrix to demonstrate how far each node is looking. First, we look at an example attention matrix formed by an attention head. Next, for the sake of visualization, we merge the attention matrices for different heads together by averaging and normalizing them to values between [0 1]. We do this for two different layers at different depths of the model. Note that these patterns are specific to a particular input graph – since the aggregation process is dynamic they would be different for different inputs. To make a complete analysis of each layer's attention we also plot the weights assigned at different distances averaged over all the attention heads for all the nodes and all the graphs in a dataset. Note that a convolutional aggregation of immediate neighbors would correspond to non-zero weights being assigned to only 0/1 hop.

We see that the attention matrices for individual attention heads are quite sparse. So, the nodes are selective about where to look. For the ZINC dataset, from Fig. 3 (a), at layer $\ell = 1$ we see that EGT approximately follows a convolutional pattern. But as we go deeper, the nodes start to take advantage of global self-attention to look further. Finally, at $\ell = 10$ we see highly non-local behavior. This shows why EGT has an advantage over local aggregation based convolutional networks because of its ability to aggregate global features. For PCQM4Mv2, in Fig. 3 (b), we notice such non-local aggregation patterns starting from the lowest layers. This shows why a global aggregation based model such as EGT has a clear advantage over convolutional networks (as seen in Table 2), because it would take a large number of consecutive convolutions to mimic such patterns. This non-local behavior is more subtle in TSP, where, except for the last layer, attention is mostly constrained to 1-3 hops, as seen from Fig. 3(c). This also shows why EGT-Constrained achieves good results on this dataset (Table 4). However, even the slight advantage of global aggregation gives pure EGT an edge over EGT-constrained. To conclude, the aggregation performed by our model is sparse and selective, like convolution, yet capable of being non-local and dynamic, which leads to a clear advantage over convolutional networks.

5 CONCLUSION AND FUTURE WORK

We proposed a simple extension – edge channels – to the transformer framework. We preserve the key idea, namely, global attention, while making it powerful enough to take structural information of the graph as input and also to process it and output new structural information such as new links and edge labels. One of our key findings is that the incorporation of the convolutional aggregation pattern is not an essential inductive bias for GNNs and instead the model can directly learn to make the best use of structural information. We established this claim by presenting experimental

results on both medium-scale, large-scale and transfer learning settings where our model achieves superior performance, beating convolutional GNNs. We also achieve a new state-of-the-art result on the large-scale PCQM4Mv2 molecular dataset. We demonstrated that the performance improvement is directly linked to the non-local nature of aggregation of the model. In future work, we aim to evaluate the performance of EGT in transductive, semi-supervised and unsupervised settings. Also, we plan to explore the prospect of reducing the computation and memory cost of our model to a sub-quadratic scale by incorporating linear attention [9, 23, 35] and sparse edge channels.

ACKNOWLEDGMENTS

This work was supported by the Rensselaer-IBM AI Research Collaboration, part of the IBM AI Horizons Network.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).
[2] Dominique Beani, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. 2021. Directional graph networks. In *International Conference on Machine Learning*. PMLR, 748–758.
[3] Mikhail Belkin and Partha Niyogi. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering.. In *Nips*, Vol. 14. 585–591.
[4] Xavier Bresson and Thomas Laurent. 2017. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553* (2017).
[5] Rémy Brossard, Oriel Frigo, and David Dehaene. 2020. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069* (2020).
[6] Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 7464–7471.
[7] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020. Generative pretraining from pixels. In *International Conference on Machine Learning*. PMLR, 1691–1703.
[8] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509* (2019).
[9] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, and Others. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794* (2020).
[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).
[11] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. 2019. On the relationship between self-attention and convolutional layers. *arXiv preprint arXiv:1911.03584* (2019).
[12] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718* (2020).
[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
[14] Vijay Prakash Dwivedi and Xavier Bresson. 2020. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699* (2020).
[15] Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2020. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982* (2020).
[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*. PMLR, 1263–1272.
[17] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, Vol. 2. IEEE, 729–734.
[18] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).
[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[20] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430* (2021).
[21] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687* (2020).
[22] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*. 2704–2710.
[23] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*. PMLR, 5156–5165.
[24] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[25] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. 2020. Flag: Adversarial data augmentation for graph neural networks. *arXiv preprint arXiv:2010.09891* (2020).
[26] Tuan Le, Marco Bertolini, Frank Noé, and Djork-Arné Clevert. 2021. Parameterized hypercomplex graph neural networks for graph classification. In *International Conference on Artificial Neural Networks*. Springer, 204–216.
[27] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739* (2020).
[28] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. 2019. Neural speech synthesis with transformer network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 6706–6713.
[29] Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. 2019. Relational pooling for graph representations. In *International Conference on Machine Learning*. PMLR, 4663–4673.
[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
[31] Omri Puny, Heli Ben-Hamu, and Yaron Lipman. 2020. Global Attention Improves Graph Networks Generalization. *arXiv preprint arXiv:2006.07846* (2020).
[32] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. 2019. Stand-alone self-attention in vision models. *arXiv preprint arXiv:1906.05909* (2019).
[33] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. 2020. Self-supervised graph transformer on large-scale molecular data. *arXiv preprint arXiv:2007.02835* (2020).
[34] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.
[35] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight memory systems. *arXiv preprint arXiv:2102.11174* (2021).
[36] Balasubramaniam Srinivasan and Bruno Ribeiro. 2019. On the equivalence between positional node embeddings and structural graph representations. *arXiv preprint arXiv:1910.00452* (2019).
[37] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.
[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.
[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).
[40] Chen Wang and Chengyuan Deng. 2021. On the Global Self-attention Mechanism for Graph Convolutional Networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 8531–8538.
[41] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. 2020. On layer normalization in the transformer architecture. In *International Conference on Machine Learning*. PMLR, 10524–10533.
[42] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).
[43] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Bad for Graph Representation? *arXiv preprint arXiv:2106.05234* (2021).
[44] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in Neural Information Processing Systems* 32 (2019), 11983–11993.
[45] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. 2020. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140* (2020).
[46] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. 2020. Revisiting graph neural networks for link prediction. *arXiv preprint arXiv:2010.16103* (2020).

A DATA AND CODE AVALABILITY

Data: All datasets used in this work are publicly available. The medium-scale GNN benchmarking datasets by Dwivedi et al. [15] are available at https://github.com/graphdeeplearning/benchmarking-gnns. The OGB-LSC [21] PCQM4M and PCQM4Mv2 large-scale datasets, and the OGB datasets [20] MolPCBA and MolHIV are available at https://ogb.stanford.edu.
Code: The code to reproduce the results presented in this work is available at https://github.com/shamim-hussain/egt.

B TRAINING METHOD AND HYPERPARAMETERS

B.1 Medium-scale Experiments

For medium-scale experiments on the PATTERN, CLUSTER, MNIST, CIFAR10, TSP, and ZINC datasets we follow the benchmarking setting suggested by Dwivedi et al. [15] and maintain a specified parameter budget of either 100K or 500K. The number of layers, the width of the node and the edge channels (L, d_h and d_e, correspondingly) were varied to get the best results on the validation set. We used the Adam optimizer and reduce the learning rate by a factor of 0.5 if the validation loss does not improve for a given number of epochs (Reduce LR when validation loss plateaus). We keep track of the validation loss at the end of each epoch and pick the set of weights that produces the least validation loss. No dropout or weight decay is used for a fair comparison with other GNNs. Each experiment (training and evaluation) was run 4 times with 4 different random seeds and the results were used to calculate the mean and standard deviations of the metric. The common hyperparameters and methods for all datasets are given in Table 6, whereas the hyperparameters which are specific for each dataset are given in Table 7.

B.2 Large-scale Experiments

While training large models on the PCQM4M and PCQM4Mv2 datasets, we found it essential to use learning rate warmup. Following the warmup, we applied cosine decay to the learning rate.

Table 6: Common hyperparameters used in medium-scale experiments on all datasets.

Hyperparameter	Value
Number of attention heads, H	8
Node channels FFN multiplier	2
Edge channels FFN multiplier	2
Final (two) MLP layers dimension	$d_h/2, d_h/4$
Virtual nodes	Not used
SVD encoding rank, r	8
Random attention masking rate	0.1
Dynamic Centrality Scalers	Not used
Dropout	Not used
Adam: initial LR	5×10^{-4}
Adam: β_1	0.9
Adam: β_2	0.999
Adam: ϵ	10^{-7}
Reduce LR by factor	0.5
Minimum LR	5×10^{-6}
LR warmup	Not used
Cosine decay	Not used

We used virtual nodes which is a more scalable method than global average pooling because the use of multiple virtual nodes allows the model to collect more graph-level information. Instead of random masking of the attention matrices, we applied dropout to the attention matrices, which showed better regularization performance. Attention dropout is the only regularization method used for all models. We trained all models for a fixed number (1 million) of gradient update steps. The hyperparameters are shown in Table 8.

B.3 Transfer Learning Experiments

We took the $\text{EGT}_{\text{Larger}}$ model pre-trained on the PCQM4Mv2 dataset (Table 8) and fine-tuned it on the OGB datasets MolPCBA and MolHIV. We used the same learning rate and warmup and cosine decay method mentioned above, although for a smaller number of total gradient update steps. Hyperparameters specific to the fine-tuning stage are shown in Table 9. Other hyper hyperparameters were the same as in Table 8. Each experiment (training and evaluation) was run 4 times with 4 different random seeds and the results were used to calculate the mean and standard deviations of the metric.

C DETAILS OF ABLATED VARIANTS

For the ablation study presented in section 4.4 of the paper, we discuss here different ablation methods in more detail.
EGT-Simple: EGT-simple uses global self-attention, but does not have dedicated residual channels for updating pairwise information (edges). The input edge embeddings (formed from graph structural matrix and edge features) directly participate in the aggregation process as follows:

$$\tilde{\mathbf{A}}^{k,\ell} = \text{softmax}(\hat{\mathbf{H}}^{k,\ell}) \odot \sigma(\mathbf{G}_0^{k,\ell}) \tag{12}$$

$$\text{Where,} \quad \hat{\mathbf{H}}^{k,\ell} = \text{clip}\left(\frac{\mathbf{Q}_h^{k,\ell}(\mathbf{K}_h^{k,\ell})^T}{\sqrt{d_k}}\right) + \mathbf{E}_0^{k,\ell} \tag{13}$$

$\mathbf{E}_0^{k,\ell}, \mathbf{G}_0^{k,\ell} \in \mathbb{R}^{N\times N}$ are directly formed by concatenations of the learned linear transformed input edge embeddings, i.e., $\mathbf{E}^{k,\ell}e_{ij}^0$, $\mathbf{G}^{k,\ell}e_{ij}^0$ respectively. Also, dynamic centrality scalers are derived from e_{ij}^0. The absence of edge channels means that the edge embeddings e_{ij} are not updated from layer to layer. So, edge classification is performed from pairwise node embeddings and input edge features. We denote this variant as EGT-Simple since it is architecturally simpler than EGT.

We use the same hyperparameters for this variant as the ones used for original EGT (Table 7, Table 8; d_e denotes only the dimensionality of the input edge embeddings) except, $d_h = 64$, $d_e = 8$ for CIFAR10, and $d_h = 80$, $d_e = 8$ for ZINC are used to make the number of parameters comparable.
EGT-Constrained: EGT-Constrained is a convolutional variant of EGT which limits the self-attention process to the 1-hop neighborhood of each node. It only keeps track of the edge embeddings e_{ij} in the edge channels if there is an edge from node i to node j or $i = j$ (self-loop). So, pairwise information corresponding to only the existing edges is updated by the edge channels. This architecture can be derived by taking the softmax over $j \in \mathcal{N}(i) \cup \{i\}$ while calculating the attention weights $\tilde{\mathbf{A}}_{ij}^{k,\ell}$ and limiting the aggregation

Table 7: Specific hyperparameters used for each dataset in medium-scale experiments. $\mathbf{D}^{(16)}$ is the distance matrix clipped to 16 hops. A is the adjacency matrix with self-loops. Distance prediction objective is only used for MNIST, CIFAR10 and ZINC datasets.

	PATTERN		CLUSTER	MNIST	CIFAR10	TSP		ZINC	
Hyperparameter	**#Param ≈100K**	**#Param ≈500K**	**#Param ≈500K**	**#Param ≈100K**	**#Param ≈100K**	**#Param ≈100K**	**#Param ≈500K**	**#Param ≈100K**	**#Param ≈500K**
Input structural matrix	$\mathbf{D}^{(16)}$	$\mathbf{D}^{(16)}$	$\mathbf{D}^{(16)}$	**A**	**A**	$\mathbf{D}^{(16)}$	$\mathbf{D}^{(16)}$	**A**	**A**
Batch size	128	128	128	128	128	8	8	128	128
Maximum no. of epochs	200	200	200	200	200	200	200	600	600
Reduce LR patience (epochs)	10	10	10	10	10	5	5	20	20
Distance prediction objective: ν (when used)				3 hops	3 hops			3 hops	3 hops
Distance prediction objective: κ (when used)				5×10^{-4}	5×10^{-4}			5×10^{-2}	5×10^{-2}
Number of layers, L	4	16	16	4	4	4	16	4	10
Node channels width, d_h	64	64	64	64	48	64	64	48	64
Edge channels width, d_e	8	8	8	8	48	8	8	48	64

Table 8: Hyperparameters used in large-scale experiments.

Hyperparameter	Value
Input structural matrix	Distance matrix (clipped up to 16 hops)
Number of attention heads, H	32
Edge channels width, d_e	64
Node channels FFN multiplier	1
Edge channels FFN multiplier	1
Final (two) MLP layers dimension	d_h, d_h
Virtual nodes	4
Dynamic Centrality Scalers	Used
SVD encoding rank, r	8
Distance prediction objective	Not used
Random attention masking	Not used
Attention matrix dropout rate	0.3
Adam: β_1	0.9
Adam: β_2	0.999
Adam: ϵ	10^{-7}
Reduce LR on loss plateau	Not used
Minimum LR	1×10^{-6}
Batch size	512
LR warmup	200,000 steps
Cosine decay	800,000 steps
Specific to $\mathbf{EGT_{Small}}$	
Maximum LR	2×10^{-4}
Number of layers, L	6
Node channels width, d_h	512
Specific to $\mathbf{EGT_{Medium}}$	
Maximum LR	1×10^{-4}
Number of layers, L	18
Node channels width, d_h	640
Specific to $\mathbf{EGT_{Large}}$	
Maximum LR	1×10^{-4}
Number of layers, L	24
Node channels width, d_h	768
Specific to $\mathbf{EGT_{Larger}}$	
Maximum LR	8×10^{-5}
Number of layers, L	30
Node channels width, d_h	768

process to neighbors as:

$$\hat{\hat{h}}_i^\ell = h_i^{\ell-1} + \mathbf{O}_h^\ell \Big\|_{k=1}^{H} \sum_{j \in \mathcal{N}(i) \cup \{i\}} \tilde{\mathbf{A}}_{ij}^{k,\ell} (\mathbf{V}^{k,\ell} \hat{h}_i^\ell) \tag{14}$$

Since this architecture is constrained to the existing edges we denote this as EGT-Constrained. It has the advantage that depending on the sparsity of the graph, it can have sub-quadratic computational and memory costs. However, it can be difficult to perform sparse aggregation in parallel on the GPU. Instead of sparse operations, we used masked attention to implement this architecture for faster training on datasets containing smaller graphs because we can take advantage of highly parallel tensor operations.

For this variant, d_h, d_e bear their usual meanings in the hyperparameters tables (Table 7, Table 8). We use the same hyperparameters for this variant as the ones used for original EGT.

Ungated Variant: In EGT, the edge channels participate in the aggregation process in two ways - by an attention bias and also by gating the values before they are aggregated by the attention mechanism. To verify the utility of the gating mechanism used in EGT, an ungated variant can be formulated by simplifying the aggregation process as follows:

$$\tilde{\mathbf{A}}^{k,\ell} = \text{softmax}(\hat{\mathbf{H}}^{k,\ell}) \tag{15}$$

$$\text{Where, } \hat{\mathbf{H}}^{k,\ell} = \text{clip}\left(\frac{\mathbf{Q}_h^{k,\ell}(\mathbf{K}_h^{k,\ell})^T}{\sqrt{d_k}}\right) + \mathbf{E}_e^{k,\ell} \tag{16}$$

$\mathbf{E}_e^{k,\ell} \in \mathbb{R}^{N \times N}$ is a concatenation of the learned linear transformed edge embeddings, i.e., $\mathbf{E}^{k,\ell}\hat{e}_{ij}^\ell$. Note that we omitted the sigmoid gates. The edge channels influence the aggregation process only via an attention bias.

Table 9: Hyperparameters used in transfer learning experiments.

Hyperparameter	MolPCBA	MolHIV
Maximum LR	1×10^{-4}	1×10^{-4}
Minimum LR	1×10^{-6}	1×10^{-6}
Batch size	16	12
LR warmup	20,000 steps	1,000 steps
Cosine decay	180,000 steps	2,000 steps