Representing Long-Range Context for Graph Neural Networks with Global Attention

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph neural networks are powerful architectures for structured datasets. However, current methods struggle to represent long-range dependencies. Scaling the depth or width of GNNs is insufficient to broaden receptive fields as larger GNNs encounter optimization instabilities such as vanishing gradients and representation oversmoothing, while pooling-based approaches have yet to become as universally useful as in computer vision. In this work, we propose the use of Transformer-based self-attention to learn long-range pairwise relationships, with a novel "readout" mechanism to obtain a global graph embedding. Inspired by recent computer vision results that finds position-invariant attention performant in learning long-range relationships, our method, which we call `GraphTrans`, applies a permutation-invariant Transformer module after a standard GNN module. This simple architecture leads to state-of-the-art results on several graph classification tasks, outperforming methods that explicitly encode graph structure. Our results suggest that purely-learning-based approaches without graph structure may be suitable for learning high-level, long-range relationships on graphs.

1 Introduction

Graph neural networks (GNNs) enable deep networks to process structured inputs such as molecules or social networks. GNNs learn mappings that compute representations at graph nodes and/or edges from the structure of and features in their neighborhoods. This neighborhood-local aggregation leverages the relational inductive bias encoded by the graph's connectivity [2]. Similar to convolutional neural networks (CNNs), GNNs can aggregate information from beyond local neighborhoods by stacking layers, effectively broadening the GNN receptive field.

However, it has been observed that GNN performance drops dramatically when a GNN's depth increases past a handful of layers. This limitation has hurt the performance of GNNs on whole-graph classification and regression tasks, where we want to predict a target value describing the whole graph that may rely on long-range dependencies that may not be captured by a GNN with a limited receptive field [32]. Consider for example a large graph where node A must attend to a distant node B which is K-hops away. If our GNN layer aggregates only over a node's one-hop neighborhood, then a K-layer GNN is required. However, the width of the receptive field of this GNN will grow exponentially, diluting the signal from node B. That is, simply expanding the receptive field to a K-hop neighborhood may not capture these long-range dependencies either [37]. Often, "too deep" GNNs lead to node representations that collapse to be equivalent over the entire graph, a phenomenon sometimes called *oversmoothing* or *oversquashing* [19, 4, 1]. To summarize, the maximum context size for common GNN architectures is effectively limited.

Many authors have proposed combating the oversmoothing problem via intermediate pooling operations similar to those found in today's CNNs. Graph pooling operations gradually coarsen the graph

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

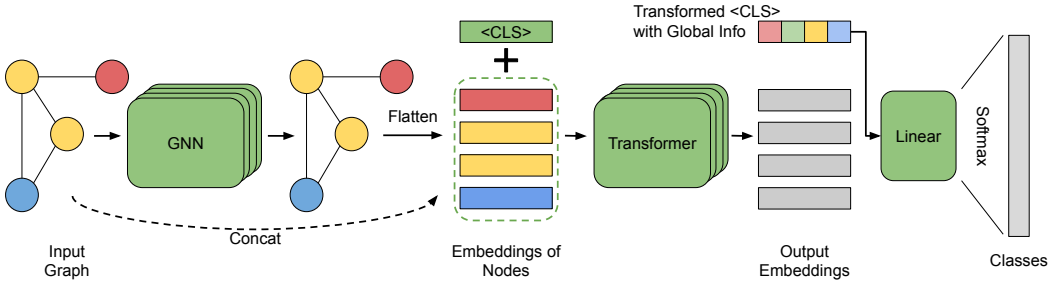


Figure 1: Architecture of `GraphTrans`. A standard GNN submodule learns local, short-range structure, then a global Transformer submodule learns global, long-range relationships.

in progressive GNN layers, usually by collapsing neighborhoods into single nodes [7, 34, 18, etc.]. In theory, hierarchical coarsening should allow better long-range learning, both by reducing the distance information has to travel and by filtering out unimportant nodes. However, as of now no graph pooling operation has been found that is as universally applicable as CNN pooling. State-of-the-art results are often obtained with models using no intermediate graph coarsening [25], and some results suggest neighborhood-local coarsening may be unnecessary or counterproductive [21].

In this work, we take a different approach at graph pooling and learning long-range dependencies in GNNs. Like hierarchical pooling, our method is also inspired by methods for computer vision: we replace some of the atomic operations that explicitly encode relevant relational inductive biases (i.e., convolutions or spatial pooling in CNNs, neighborhood coarsening in GNNs) with purely learned operations like attention [9, 3, 5].

Our method, which we call Graph Transformer (`GraphTrans`, see Fig. 1), adds a Transformer subnetwork on top of a standard GNN layer stack. This Transformer subnetwork explicitly computes all pairwise node interactions in a position-agnostic fashion. This approach is intuitive as it retains the GNN as a specialized architecture to learn local representations of the structure of a node's immediate neighborhood while leveraging the Transformer as a powerful global reasoning module. This parallels recent computer vision architectures, where authors have found hard relational inductive biases important for learning short-range patterns but less useful or even counterproductive in modeling long-range dependencies [23]. As the Transformer without a positional encoding is permutation-invariant, we find it is a natural fit for graphs. Moreover, `GraphTrans` does not require any specialized modules or architectures and can be implemented in any framework atop any existing GNN backbone.

We evaluate `GraphTrans` on a variety of popular graph classification datasets. We find significant improvements in accuracy on OpenGraphBenchmark [13] where we achieve state-of-the-art results on two graph classification tasks. Moreover, we find substantial improvements on the molecular dataset NCI1. Surprisingly, we find our simple model outperforms complex baselines for long-range modeling in graphs via hierarchical clustering such as self-attention pooling [18].

Our contributions are as follows:

- We show that long-range reasoning via Transformers improve graph neural network (GNN) accuracy. Our results suggest that modelling all pairwise node-node interactions in the graph is particularly important for large graph classification tasks.

- We introduce a novel GNN "readout module." Inspired by text-classification applications of Transformers, we use a special "`<CLS>`" token whose output embedding aggregates all pairwise interactions into a single classification vector. We find that this approach outperforms both non-learned readout methods like global pooling as well as learned aggregation methods like graph-specific pooling methods [34, 18] and "virtual node" approaches.

- Using our novel architecture `GraphTrans`, we obtain state-of-the-art results on several OpenGraph-Benchmark [13] datasets and the NCI biomolecular datasets [28].

2 Related work

Graph Pooling. Similar to CNNs, pooling in GNNs can be either global, reducing a set of node and/or edge encodings to a single graph encoding, or local, collapsing subsets of nodes and/or edges to create a coarser graph. Paralleling the use of intermediate pooling within CNNs, several authors have proposed local pooling operations meant to be used within the GNN layer stack, progressively coarsening the graph. Methods proposed include both learned pooling schemes [34, 18, 12, 14, 15, etc.] and non-learned pooling methods based on classic graph coarsening schemes [8, 7, etc.]. However, the effectiveness or necessity of hierarchical, coarsening-based pooling in GNNs is unclear [21]. On the other hand, the most common global, whole-graph pooling methods, are i) non-learned mean or max pooling over nodes and ii) the "virtual node" approach, where a final GNN layer outputs an embedding for a single virtual node that is connected to every "real" node in the graph.

A notable work related to graph pooling is the DAGNN (Directed Acyclic Graph Neural Network) of Thost and Chen [25], which had obtained the previous state-of-the-art accuracy on OGBG-Code2. The DAGNN layer aggregates over the entire graph within each layer via an RNN that traverses the DAG, unlike most GNN layers that only aggregate over a node's neighborhood. While they did not characterize this method as a pooling operation, it is similar to `GraphTrans` in that it acts as a learned global pooling (in that it aggregates the embeddings of every node in a DAG into the sink nodes) that can model long-range dependencies. Note that `GraphTrans` is also complementary to DAGNN because their final graph-level pooling operation is a global max-pooling over the sink nodes rather than a learned operation.

Transformers on Graphs. Several authors have investigated applications of Transformer architectures to graphs. Recent works such as Zhang et al. [35], Rong et al. [22], and Dwivedi and Bresson [10] propose GNN layers that let nodes attend to other nodes in some surrounding neighborhood via Transformer-style attention, whereas we use self attention for a permutation-invariant, graph-level pooling or "readout" operation that collapses node encodings to a single graph encoding. Of these, Zhang et al. [35] and Rong et al. [22] tackle the problem of learning long-range dependencies without oversmoothing by allowing nodes to attend to more than just the one-hop neighborhood: Zhang et al. [35] take the attended neighborhood radius as a tuning parameter and Rong et al. [22] attend to neighborhoods of random size during training and inference. In contrast, we use whole-graph self-attention to allow for learning of long-range dependencies.

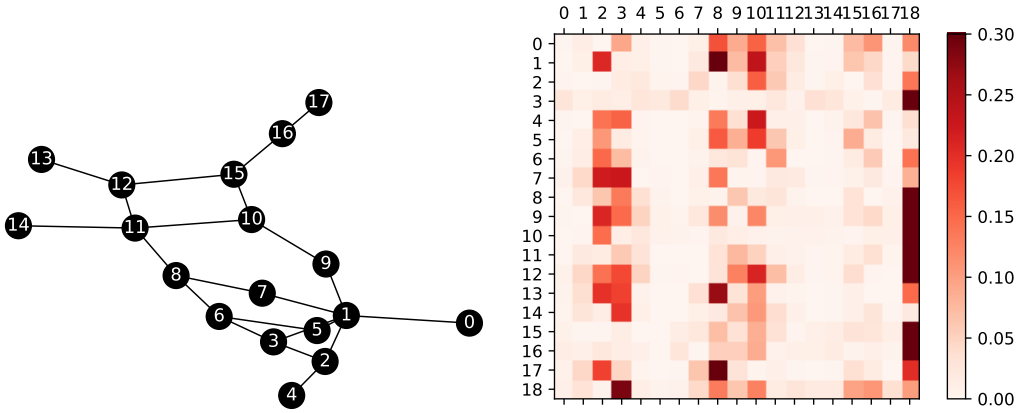
While Zhang et al. [35] do not consider whole-graph prediction problems, in the case of Dwivedi and Bresson [10], when a graph-wide embedding was needed for graph classification or regression, they used global average pooling over the nodes, while Rong et al. [22] take a weighted sum over nodes with the weights computed by passing the $\boldsymbol{h}_v^L$'s to a two-layer MLP. Note also that prior works consider graph-specific versions of a Transformer's positional encoding, while we omit positional encodings to ensure permutation invariance.

3 Motivation: Learning to model long-range pairwise interactions in gr

To summarize, attempting long-range learning on graphs via stacking GNN layers or hierarchical pooling have not yet led to performance increases, and while some works have shown some success in expanding the receptive field of a single GNN layer beyond a one-hop neighborhood [35, 22, 37], it remains to be seen how this approach will scale to very large graphs with hundreds or thousands of nodes.

An inspiration for an alternative approach can be found in the recent computer vision literature. In the last few years, researchers have found that attention mechanisms can act as drop-in replacements for traditional CNN convolutions [3, 5]: attention layers can learn to reproduce the strong relational inductive biases induced by local convolutions. More recently, state-of-the-art approaches to several computer vision tasks use an attention-style submodule on top of a traditional CNN backbone [1, 30, etc.]. These results suggest that while strong relational inductive biases are helpful for learning local, short-range correlations, for long-range correlations less structured modules may be preferred [1].

We leverage this insight to the graph learning domain with our `GraphTrans` model, which uses a traditional GNN subnetwork as a backbone, but leaves learning long-range dependencies to a Transformer subnetwork with no graph spatial priors. As mentioned, our Transformer application lets



(a) Example graph from Code2.

(b) Corresponding attention map from `GraphTrans`.

Figure 2: Example graph and attention map in our `GraphTrans`. The graph is randomly sampled from the Code2 validation set. The attention map is retrieved from the first layer of the transformer module in our `GraphTrans`. The horizontal axis corresponds to targets and the horizontal axis corresponds to sources (so, attention weights will sum to one over the horizontal axis). Note that in (b), index 18 corresponds to the special <CLS> token described in section 4.

every node attend to every other node (unlike other approaches of applying Transformers to graphs that only allow attention to neighborhoods), which incentivizes the Transformer to learn the *most important* node-node relationships, instead of favoring nearby nodes (the latter task having been offloaded to the preceding GNN module).

Qualitatively, this scheme provides evidence that long-range relationships are indeed important. An example application of `GraphTrans` on the OGB Code2 dataset is depicted in Figure 2. In this task, we take in the Abstract Sentence Tree obtained by parsing a Python method and need to predict the tokens that form the method name. The attention map exhibits similar patterns to those found in NLP applications of Transformers: some nodes receive significant weighting from many other nodes, regardless of the distance between them. Note that node 17 assigns significant importance to node 8, despite these two nodes being five hops away. Also, in Figure 2's attention map, index 18 refers to the embedding corresponding to the special `<CLS>` token we use as a readout mechanism, described in more detail below. We allow this embedding to be learnable, so the many nodes attending to it (represented by the many dark cells in column 18) may suggest these nodes are obtaining some graph-general memory from the learned embedding. This qualitative visualization, along with our new state-of-the-art results, suggest that removing spatial priors when learning long-range dependencies may be necessary for effective graph summarization.

The implementation details of `GraphTrans` are discussed next.

4 Learning Global Information with `GraphTrans`

Referring back to Figure 1, `GraphTrans` consists of two primary modules: a GNN subnetwork followed by a Transformer subnetwork. We discuss these in detail next.

GNN module. We consider graph property prediction, i.e., for each graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ we have a graph-specific prediction target $y_{\mathcal{G}}$. We suppose that each node $v \in \mathcal{V}$ has an initial feature vector $\boldsymbol{h}_v^0 \in \mathbb{R}^{d_0}$. As `GraphTrans` is a generally-applicable framework that can be used in concert with a variety of GNNs, we make very few assumptions on the GNN layers that feed into the Transformer subnetwork. A generic GNN layer stack can be expressed as

$$\boldsymbol{h}_v^\ell = f_\ell\left(\boldsymbol{h}_v^{\ell-1}, \{\boldsymbol{h}_u^{\ell-1} | u \in \mathcal{N}(v)\}\right), \quad \ell = 1, \ldots, L_{\text{GNN}} \tag{1}$$

where L_{GNN} is the total number of GNN layers, $\mathcal{N}(v) \subseteq \mathcal{V}$ is some neighborhood of v, and $f_\ell(\cdot)$ is some function parameterized by a neural network. Note that many GNN layers admit edge features, but to avoid notational clutter we omit discussion of them here.

Transformer module. Once we have the final per-node GNN encodings $\boldsymbol{h}_v^{L_{\text{GNN}}}$, we pass these to `GraphTrans`'s Transformer subnetwork. The Transformer subnetwork operates as follows. We first perform a linear projection of the $\boldsymbol{h}_v^{L_{\text{GNN}}}$'s to the Transformer dimension and a Layer Normalization to normalize the embedding:

$$\bar{\boldsymbol{h}}_v^0 = \text{LayerNorm}(\boldsymbol{W}^{\text{Proj}}\boldsymbol{h}_v^{L_{\text{GNN}}}) \tag{2}$$

where $\boldsymbol{W}^{\text{Proj}} \in \mathbb{R}^{d_{\text{TF}} \times d_{L_{\text{GNN}}}}$ is a learnable weight matrix, and d_{TF} and $d_{L_{\text{GNN}}}$ are the Transformer dimension and the dimension of the final GNN embedding, respectively. The projected node embeddings $\bar{\boldsymbol{h}}_v^0$ are then fed into a standard Transformer layer stack, with no additive positional embeddings, as we expect the GNN to have already encoded the structural information into the node embeddings:

$$\begin{gathered} a_{v,u}^{\ell} = (\boldsymbol{W}_{\ell}^{Q}\bar{\boldsymbol{h}}_v^{\ell-1})^{\top}(\boldsymbol{W}_{\ell}^{K}\bar{\boldsymbol{h}}_u^{\ell-1})/\sqrt{d_{\text{TF}}} \qquad \alpha_{v,u}^{\ell} = \underset{w \in \mathcal{V}}{\text{softmax}}(a_{v,w}^{\ell}) \\ \bar{\boldsymbol{h}}_v'^{\ell} = \sum_{w \in \mathcal{V}} \alpha_{v,w}^{\ell}\boldsymbol{W}_{\ell}^{V}\bar{\boldsymbol{h}}_w^{\ell-1} \end{gathered} \tag{3}$$

where $\boldsymbol{W}_{\ell}^{Q}, \boldsymbol{W}_{\ell}^{K}, \boldsymbol{W}_{\ell}^{V} \in \mathbb{R}^{d_{\text{TF}}/n_{\text{head}} \times d_{\text{TF}}/n_{\text{head}}}$ are the learned query, key, and value matrices, respectively, for a single attention head in layer ℓ. As is standard, we run $n_h ead$ parallel attention heads and concatenate the resulting per-head encodings $\bar{\boldsymbol{h}}_v'^{\ell}$. These concatenated encodings are then passed to a Transformer fully-connected subnetwork, consisting of the standard Dropout -> Layer Norm -> Fully-Connected -> nonlinearity -> Dropout -> Fully-Connected -> Dropout -> Layer Norm sequence, with residual connections from $\bar{\boldsymbol{h}}_v^{\ell-1}$ to after the first dropout, and from before the first fully-connected sublayer to after the dropout immediately following the second fully-connected sublayer.

`<CLS>` **embedding as a GNN "readout" method.** As mentioned, for whole-graph classification we require a single embedding vector that describes the whole graph. In the GNN literature, this module that collapses embeddings for every node and/or edge to a single embedding is called the "readout" module, and the most common readout modules are simple mean or max pooling, or a single "virtual node" that is connected to every other node in the network.

In this work, we propose a special-token readout module similar to those used in other applications of Transformers. In text classification tasks with Transformers, a common practice is to append a special `<CLS>` token to the input sequence before passing it into the network, then to take the output embedding corresponding to this token's position as the representation of the whole sentence. In that way, the Transformer will be trained to aggregate information of the sentence to that embedding, by calculating the one-to-one relationships between the `<CLS>` token and each other tokens in the sentence with the attention module.

Our application of special-token readout is similar to this. Concretely, when feeding the transformed per-node embeddings $\bar{h}_v^0$, we append an additional learnable embedding $h_{\texttt{<CLS>}}$ to the sequence, and take the first embedding $\bar{h}_{\texttt{<CLS>}} \in \mathbb{R}^{d_{\text{TF}}}$ from the transformer output as the representation of the whole graph (note that since we do not include positional encodings, placing the special token at the "beginning" of the sentence has no special computational meaning; the location is chosen by convention). Finally, we apply a linear projection followed by a softmax to the embedding to generate the prediction:

$$y = \text{softmax}(\boldsymbol{W}^{\text{out}}\bar{\boldsymbol{h}}_{\texttt{<CLS>}}^{L_{\text{TF}}}). \tag{4}$$

where L_{TF} is the number of Transformer layers.

This special-token readout mechanism may be viewed as a generalization or a "deep" version of a virtual node readout. While a virtual node method requires every node in the graph to send its information to the virtual node and does not allow for learning pairwise relationships between graph nodes except within the virtual node's embedding (possibly creating an information bottleneck), a Transformer-style special-token readout method lets the network learn long-range node-to-node relationships in earlier layers before needing to distill them in the later layers.

5 Experiments

We evaluate `GraphTrans` on graph classification tasks from three modalities: biology, computer programming, and chemistry. Our `GraphTrans` achieves consistent improvement over all of these

Model	GNN Type	#GNN Layer	NCI1	NCI109
Set2Set [27, 18]	GCN	3	68.6± 1.9	69.8±1.2
SortPool [36, 18]	GCN	3	73.8±1.0	74.0±1.2
SAGPool$_h$ [18]	GCN	3	67.5±1.1	67.9±1.4
SAGPool$_g$ [18]	GCN	3	74.2±1.2	74.1±0.8
Strong Baseline [11, 1]	GIN	8	80.0±1.4	–
+ FA layer [1]	GIN	8	81.5±1.2	–
Transformer [26]	–	–	66.5±2.8	67.1± 2.8
`GraphTrans` (small)	GCN	3	**80.2±1.9**	**79.0±2.5**
`GraphTrans` (large)	GIN	4	**83.0±1.6**	**82.5±2.0**

Table 1: Test-set accuracies on the NCI biological datasets. Our `GraphTrans` outperforms the state-of-the-art strong baseline wih FA layer.

benchmarks, indicating the generality and effectiveness of the framework. All of our model are trained with the Adam optimizer [16] with a learning rate of 0.0001, a weight decay of 0.0001 and the default Adam β parameters. All Transformer modules used in the our experiments have an embedding dimension d_{TF} of 128 and a hidden dimension of 512 in the feedforward subnetwork. The Transformer baselines described below are trained with only the sequence of node embeddings, discarding the graph structure.

5.1 Biological Benchmarks

Datasets. We choose two commonly used graph classification benchmarks, NCI1 and NCI109 [29]. Each of them contains about 4000 graphs with around 30 nodes on average, representing biochemical compounds. The task is to predict whether a compound contains anti-lung-cancer activity. We follow the settings in [18, 1] for the NCI1 and NCI109, randomly splitting the dataset into training, validation and test set by a ratio of 8:1:1.

Training Setup. We trained `GraphTrans` on both the NCI1 and NCI109 datasets for 100 epochs with a batch size of 256. We run each experiment 20 times with different random seeds and calculate the average and standard deviation of the test accuracies. All the model follows the architecture in Figure 1, with 4 transformer layers and a dropout ratio of 0.1 for both the GNN and Transformer modules. We use two different settings adopted from prior literature for the width and depth of the GNN submodule in `GraphTrans`. The GNN module width and depth in the small `GraphTrans` model are copied from the simple baseline, i.e. the settings in [18], which has a hidden dimension of 128 and 3 GNN layers. The settings of the GNN module in the large `GraphTrans` model are adopted from the default GCN/GIN model provided by OGB, which has a hidden dimension of 300 and 4 GNN layers. We also adopt a cosine annealing schedule [20] for learning rate decay.

Results. We report the results on both NCI1 and NCI109 in Table 1. The simple baselines, including GCN Set2Set, SortPool, and SAGPool, are taken from [18], while the strong baselines as well as the FA layer comes from [1]. In Table 1, Our Graph Transformer (1) has the same architecture as the simple baseline, but improves the average accuracy by 6.0%. We also tested the framework with GIN as the encoder (`GraphTrans` (large)) to align with the settings in strong baseline, which also significantly improves the accuracy of the strong baseline by 1.5%, even without the deep GNN.

5.2 Chemical Benchmarks

Datasets. For chemical benchmarks, we evaluate our `GraphTrans` on a dataset larger than NCI dataset, `molpcba` from the Open Graph Benchmark (OGB) [13]. It contains 437929 graphs with 28 nodes on average. Each graph in the dataset represents a molecule, where nodes and edges are atoms and chemical bonds, respectively. The task is to predict multiple properties of a molecule. We use the standard splitting from the benchmark. The performance on the GIN and GIN-Virtual baselines are as reported on the OGB leaderboard [13].

Model	Valid	Test
GIN [33]	0.2305	0.2266
GIN-Virtual [33]	0.2798	0.2703
Transformer [26]	0.1347	0.1304
`GraphTrans` (GIN)	**0.2876**	**0.2721**
`GraphTrans` (GIN-Virtual)	**0.2858**	**0.2815**

Table 2: Results on OGBG-Molpcba dataset. All the results are ROC-AUC on test dataset. The models in the parenthesis indicate the GNN type of the GNN module in `GraphTrans`.

Model	Valid	Test
GCN [17]	0.1399	0.1507
GCN-Virtual [17]	0.1461	0.1629
PNA [6]	0.1442	0.1585
DAGNN (SOTA) [25]	0.1607	0.1751
Transformer [26]	0.1539	0.1660
`GraphTrans` (GCN)	0.1594	0.1748
`GraphTrans` (GCN-Virtual)	**0.1670**	**0.1810**
`GraphTrans` (PNA)	**0.1698**	**0.1819**

Table 3: Results on OGBG-Code2 dataset. All the baselines are collected from the OGB leaderboard. `GraphTrans` outperforms the state-of-the-art DAGNN. The improvement based on PNA model indicates that our method is orthogonal to the type of GNN module.

Training Setups. All the GNN modules in the experiments follows the settings of default GIN model provided in OGB, with 4 layers and 300 hidden dimension. We train all the models for 100 epochs with a batch size of 256 and report the test result with the best validation ROC-AUC. For both GNN and Transformer modules, we apply a dropout of 0.3. We use GIN as the baseline and the GNN module, since it performs better than GCN models on Molpcba dataset, according to the leaderboard.

Results. In Table 2, we report the ROC-AUC on validation and test set of Molpcba. Though Transformer along works very bad on this dataset, our `GraphTrans` still improves the ROC-AUC of the GIN and GIN-Virtual baseline. It indicates that our design could take the benefit from both the local graph structure learnt by the GNN and the long-range concept retrieved by the Transformer module based on the GNN embeddings.

5.3 Computer Programming Benchmarks

Datasets. For computer programming benchmark, we also adopt a large dataset, `code2` from OGB, which has 45741 graphs each with 125 nodes on average. The dataset is a collection of Abstract Syntax Trees (ASTs) from about 450k Python method definitions. The task is to predict the sub-tokens forming the method name, given the method body represented by the AST. We also adopt the standard dataset splitting from the benchmark. All baseline performances are as reported on the OGB leaderboard.

Training Setups. We also apply the default settings of GCN for Code2 from OGB, with 4 GNN layers, 300 hidden dimension and a dropout ratio of 0.0. We apply a dropout ratio of 0.3 to the Transformer module to avoid overfitting. We train all the models for 30 epochs with a batch size of 16, due to the large scale of the dataset. For the `GraphTrans` (PNA) model, we follow the settings in [24], with a hidden embedding of 272 for GNN module and a weight decay of 3e-6. The only difference is that we still use the learning rate of 0.0001, instead of the heavily tuned 0.00063096 [24].

Results. In Table 3, we compare our `GraphTrans` with top tier architectures on the leaderboard on Code2 dataset. As the average number of nodes in each graph increases, the global information becomes more important as it becomes more difficult for the GNN to gather information from nodes

Model	Valid	Test
Pretrained GCN-Virtual	0.1457	0.1574
Graph Transformer w./ pretrained GCN-Virtual (freeze)	0.1479	0.1616
Graph Transformer w./ pretrained GCN-Virtual (non-freeze)	**0.1564**	**0.1733**

Table 4: Ablation studies on OGBG-Code2 dataset. Only training the Transformer module in `GraphTrans` with pretrained/freezed GNN module also improves the F1-score. It indicates that the transformer training on GNN-encoded embeddings can learn information that hard to be captured by the GNN along.

Model	Valid	Test
Graph Transformer - mean	0.1398	0.1509
Graph Transformer - last	0.1566	0.1716
Graph Transformer - <cls>	0.1593	0.1784
Graph Transformer - <cls> - cat	**0.1670**	**0.1810**

Table 5: Ablation studies for <cls> token and feature concatenation for our Graph Transformer. The `mean` and `last` are two commonly used embedding aggregation method for sequence classification.

far away. Even without heavy tuning, `GraphTrans` significantly outperforms the state-of-the-art (DAGNN) [25] on the leaderboard.

We also include the results for PNA model and our `GraphTrans` with PNA model as the GNN encoder. Our `GraphTrans` also significantly improves the result, which indicates that our architecture is orthogonal to the variants of GNN encoder module.

5.4 Transformers can Capture Long-Range Information in Graphs.

As we previously observed in Figure 2 and discussed in Section 3, the attention inside the transformer module can capture long-range information that are hard to be learned by the GNN module.

To further verify the hypothesis, we designed an experiment to show that the Transformer module can learn additional information to the GNN module. In Table 4, we first pretrain a GNN (GCN-Virtual) until converge on the Code2 dataset, and then freeze the GNN model and plug our Transformer module after it. By training the model on the training set with fixed GNN module, we can still observe a 0.0022 F1-score improvement on the validation set and 0.0042 on the test set. It indicates that the Transformer can learn additional information that is hard to be learnt by GNN module along.

With pretrained/non-freeze GNN module, our `GraphTrans` can achieve even higher F1-score. That may because the GNN module can now focus on learning the local structure information, by leaving the long-range information learning to the Transformer layer after it. The model benefits from the specialization as mentioned in [31]. Note that for all the experiments in Table 4, we do not concatenate the embeddings from the input graph to input of Transformer for simplicity.

5.5 Effectiveness of `<CLS>` Embedding

In Figure 2b, we can observe that the row 18 (the last row is for `<CLS>`) has dark red on multiple columns, which indicates that the `<CLS>` learns to attend to important nodes in the graph in order to learn the representation for the whole graph.

We also examined the effectiveness of our `<CLS>` embedding quantitatively. In Table 5, we tested several common methods to for sequence classification. The `mean` operation averages the output embeddings of the transformer to a single graph embedding; the `last` operation takes the last embedding in the output sequence as the graph embedding. The quantitative results indicate that the `<CLS>` embedding is most effective with 0.0275 improvement on test set, as model can learn to retrieve information from different nodes and aggregate them into one embedding. The concatenation of the embeddings in the input graph and the input embeddings of the transformer can further improve the validation and test F1-score to 0.1670 and 0.1733.

6 Conclusion

We proposed `GraphTrans`, a simple yet powerful framework for learning long-range relationships with GNNs. Leveraging recent results that suggest structural priors may be unnecessary or even counterproductive for high-level, long-range relationships, we augment standard GNN layer stacks with a subsequent permutation-invariant Transformer module. The Transformer module acts as a novel GNN "readout" module, simultaneously allowing the learning of pairwise interactions between graph nodes and summarizing them into a special token's embedding as is done in common NLP applications of Transformers. This simple framework leads to surprising improvements upon the state of the art in several graph classification tasks, outperforming some methods that attempt to encode much more structural information.

References

[1] U. Alon and E. Yahav. On the Bottleneck of Graph Neural Networks and its Practical Implications. In *International Conference on Learning Representations*, Mar. 2021.

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]*, June 2018.

[3] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-End Object Detection with Transformers. *arXiv:2005.12872 [cs]*, May 2020.

[4] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and Relieving the Over-smoothing Problem for Graph Neural Networks from the Topological View. In *AAAI 2020*, pages 3438–3445, New York, NY, USA, Feb. 2020. doi: 10.1609/aaai.v34i04.5747.

[5] J.-B. Cordonnier, A. Loukas, and M. Jaggi. On the Relationship between Self-Attention and Convolutional Layers. In *International Conference on Learning Representations*, 2020.

[6] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. Principal neighbourhood aggregation for graph nets. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13260–13271. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/99cad265a1768cc2dd013f0e740300ae-Paper.pdf.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, volume 29, pages 3844–3852, 2016.

[8] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted Graph Cuts without Eigenvectors A Multilevel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, Nov. 2007. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.1115.

[9] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929 [cs]*, Oct. 2020.

[10] V. P. Dwivedi and X. Bresson. A Generalization of Transformer Networks to Graphs. *arXiv:2012.09699 [cs]*, Jan. 2021.

[11] F. Errica, M. Podda, D. Bacciu, and A. Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HygDF6NFPB.

[12] H. Gao and S. Ji. Graph U-Nets. *arXiv:1905.05178 [cs, stat]*, May 2019.

[13] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv:2005.00687 [cs, stat]*, 2020.

[14] J. Huang, Z. Li, N. Li, S. Liu, and G. Li. AttPool: Towards Hierarchical Feature Representation in Graph Convolutional Networks via Attention Mechanism. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6480–6489, 2019.

[15] A. H. Khasahmadi, K. Hassani, P. Moradi, L. Lee, and Q. Morris. Memory-Based Graph Networks. In *International Conference on Learning Representations*, 2020.

[16] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[17] T. N. Kipf et al. Keras-GCN. https://github.com/tkipf/keras-gcn, 2017.

[18] J. Lee, I. Lee, and J. Kang. Self-Attention Graph Pooling. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 3734–3743, Long Beach, CA, June 2019. PMLR.

[19] Q. Li, Z. Han, and X.-m. Wu. Deeper Insights Into Graph Convolutional Networks for Semi-Supervised Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pages 3538–3545, Apr. 2018.

[20] I. Loshchilov and F. Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. In *International Conference on Learning Representations*, May 2017.

[21] D. Mesquita, A. H. Souza, and S. Kaski. Rethinking pooling in graph neural networks. In *Advances in Neural Information Processing Systems*, volume 33, pages 2220–2231, Oct. 2020.

[22] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. WEI, W. Huang, and J. Huang. Self-supervised graph transformer on large-scale molecular data. In *Advances in Neural Information Processing Systems*, volume 33, pages 12559–12571. Curran Associates, Inc., 2020.

[23] A. Srinivas, T.-Y. Lin, N. Parmar, J. Shlens, P. Abbeel, and A. Vaswani. Bottleneck Transformers for Visual Recognition. *arXiv:2101.11605 [cs]*, Jan. 2021.

[24] S. A. Tailor, F. L. Opolka, P. Liò, and N. D. Lane. Adaptive filters and aggregator fusion for efficient graph convolutions. *arXiv preprint arXiv:2104.01481*, 2021.

[25] V. Thost and J. Chen. Directed Acyclic Graph Neural Networks. In *International Conference on Learning Representations*, 2021.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, 2017.

[27] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets, 2016.

[28] N. Wale and G. Karypis. Comparison of Descriptor Spaces for Chemical Compound Retrieval and Classification. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 678–689, Dec. 2006. doi: 10.1109/ICDM.2006.39.

[29] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, Mar. 2008. ISSN 0219-3116. doi: 10.1007/s10115-007-0103-5. URL https://doi.org/10.1007/s10115-007-0103-5.

[30] H. Wang, W. Wang, and J. Liu. Temporal Memory Attention for Video Semantic Segmentation. *arXiv:2102.08643 [cs]*, Feb. 2021.

[31] Z. Wu*, Z. Liu*, J. Lin, Y. Lin, and S. Han. Lite transformer with long-short range attention. In *International Conference on Learning Representations (ICLR)*, 2020.

[32] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, July 2018.

[33] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How Powerful are Graph Neural Networks? *arXiv:1810.00826 [cs, stat]*, Feb. 2019.

[34] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, pages 4805–4815, 2018.

[35] J. Zhang, H. Zhang, C. Xia, and L. Sun. Graph-Bert: Only Attention is Needed for Learning Graph Representations. *arXiv:2001.05140 [cs, stat]*, Jan. 2020.

[36] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[37] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7793–7804. Curran Associates, Inc., 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] The main potentially negative societal impact of our work is potentially raising the barrier to entry for GNN research. It is well-known that state-of-the-art Transformer models are among the largest and most costly deep neural networks that exist today. If future results confirm that all-to-all Transformer modules are necessary for top performance on graph datasets, this could potentially make state-of-the-art GNNs as inaccessible to the general public as state-of-the-art NLP models like GPT-3.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We report standard deviations for small datasets. For the large datasets, accuracy results are quite stable.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code is included
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]