GRPE: Relative Positional Encoding for Graph Transformer

[1,2]**Wonpyo Park***, [1]**Woonggi Chang***, [1]**Donggeon Lee**, [1]**Juntae Kim**, [2]**Seung-won Hwang**
[1]Standigm, [2]Seoul National University

Abstract

Designing an efficient model to encode graphs is a key challenge of molecular representation learning. Transformer built upon efficient self-attention is a natural choice for graph processing, but it requires explicit incorporation of positional information. Existing approaches either linearize a graph to encode absolution position in the sequence of nodes, or encode relative position with another node using bias terms. The former loses preciseness of relative position from linearization, while the latter loses a tight integration of node-edge and node-spatial information. In this work, we propose relative positional encoding for a graph to overcome the weakness of the previous approaches. Our method encodes a graph without linearization and considers both node-spatial relation and node-edge relation. We name our method Graph Relative Positional Encoding dedicated to graph representation learning. Experiments conducted on various molecular property prediction datasets show that the proposed method outperforms previous approaches significantly. Our code is publicly available at `https://github.com/lenscloth/GRPE`.

1 Introduction

The average cost of drug discovery has increased drastically in the last decade with a declining success rate of developing new therapeutics. Large-scale screening with deep neural networks draws considerable attention to lower the cost of drug discovery, especially in the lead finding and lead optimization phase. By representing a molecule in a graph, a graph neural network can be utilized on the screening by predicting important properties, *e.g.*, drug-likeness, solubility, or synthesizability. Therefore, graph representation learning has become a key technique for drug discovery.

Transformer introduced by Vaswani et al. (2017), has been effective in graph representation learning, overcoming inductive biases of graph convolutional networks by using self-attention. On the other hand, explicit representations of position in graph convolutional networks are lost, thus incorporating graph structure on the hidden representations of self-attention is a key challenge. We categorize existing works into two: (a) linearizing graph with graph Laplacian to encode the absolute position of each node (Dwivedi & Bresson, 2020; Kreuzer et al., 2021) (b) encoding position relative to another node with bias terms (Ying et al., 2021). The former loses preciseness of position due to linearization, while the latter loses a tight integration of node-edge and node-spatial information. Ours can be interpreted as overcoming the weakness of the two, which considers both relative position and the interaction with node.

To avoid losing relative position due to the linearization, we propose to adopt the relative positional encoding by Shaw et al. (2018). Its efficacy has been verified in natural language processing, however, the efficacy is not yet studied in a graph. Since the original work is designed for 1D word sequences, we reformulate to incorporate graph-specific properties. To this end, we introduce two sets of learnable positional encoding vectors to represent spatial relation or edge between two nodes. Our method considers the interaction between node features and the two encoding vectors to integrate both node-spatial relation and node-edge information. We name our method for graph representation learning as **G**raph **R**elative **P**ositional **E**ncoding (GRPE).

*Equal contribution

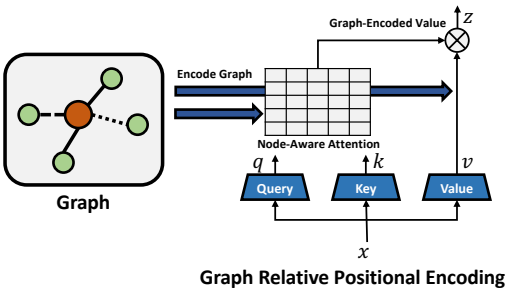

Figure 1: Our proposed Graph Relative Positional Encoding (GRPE). GRPE is a relative positional encoding method dedicated to learn graph representation.

We extensively conducted experiments to validate the efficacy of our proposed method on various tasks from graph classification to graph regression. Models built with our relative positional encoding achieve state-of-the-art performance on various molecule property prediction datasets, showing the efficacy of our proposed method.

2 Related Work

Existing works leverage Transformer architecture to learn graph representation. We categorize those methods as follows.

Earlier models adopt Transformer without explicit encoding of positional information on a graph. Veličković et al. (2017) replace graph convolution operation with self-attention module where attention is only performed within neighboring nodes. Rong et al. (2020) stack self-attention module next to the graph convolutional networks iteratively to consider long-range interaction between nodes. In their method, affinity is considered only on the graph convolutional networks, and positional information is not given on self-attention.

Later works employ absolute positional encoding to explicitly encode positional information of graph on Transformer. Their main idea is to linearize a graph into a sequence of nodes, and an absolute positional encoding is added to the input feature. Dwivedi & Bresson (2020) adopted graph Laplacian as a positional encoding, where each cell of encoding represents partitions after graph min-cut. Nodes sharing many partitions after graph min-cut would have similar graph Laplacian vectors. Kreuzer et al. (2021) employ a learnable positional encoding with a Transformer where its input is the Laplacian spectrum of a graph. Due to the linearization of a graph, those approaches lose the preciseness of position on the graph.

Meanwhile, encoding relative positional information has been studied to avoid losing the preciseness of position. Graphormer introduced by Ying et al. (2021) encodes relative position on scaled dot product attention map by adding bias terms. However, the bias terms are parameterized only relative position such as shortest path distance or edge type, and interaction with node features is lost. On the other hand, Shaw et al. (2018) introduce relative positional encoding, which considers tight integration of node and relative position, however, their method is designed to process 1D word sequences. We extend the systematic design of Shaw et al. (2018) for a graph. Our method considers the interaction between nodes and graph-specific properties such as edges, and encodes precise positional information of a graph.

3 Background

Notation. We denote a set of nodes on the graph $\{n_i\}_{i=1:N}$ and a set of edges on the graph $\{e_{ij}|j \in \mathcal{N}_i\}_{i=1:N}$, where N is the number of nodes and $\mathcal{N}_i$ is a set neighbors of a node n_i. Both n_i and e_{ij} are positive integer numbers to index the type of nodes or edges, *e.g.*, atom numbers or bond types of a molecule. $\psi(i, j)$ denotes a function encodes structural relationship between the node n_i and n_j.

3.1 TRANSFORMER

Transformer is built by stacking multiple self-attention layers. Self-attention maps a query and a set of key pairs to compute an attention map. Values are weighted summed with the weight on the attention map to output the hidden feature for the following layer.

Specifically, $x_i \in \mathbb{R}^{d_x}$ denotes the input feature of the node n_i, and $z_i \in \mathbb{R}^{d_z}$ denotes the output feature of the self-attention module. The self-attention module computes query q, key k, and value v with independent linear transformations: $W^{\text{query}} \in \mathbb{R}^{d_x \times d_z}$, $W^{\text{key}} \in \mathbb{R}^{d_x \times d_z}$ and $W^{\text{value}} \in \mathbb{R}^{d_x \times d_z}$.

$$q = W^{\text{query}} x, \quad k = W^{\text{key}} x \quad \text{and} \quad v = W^{\text{value}} x. \tag{1}$$

The attention map is computed by applying a scaled dot product between the queries and the keys.

$$a_{ij} = \frac{q_i \cdot k_j}{\sqrt{d_z}} \quad \text{and} \quad \hat{a}_{ij} = \frac{\exp(a_{ij})}{\sum_{k=1}^{N} \exp(a_{ik})}. \tag{2}$$

The self-attention module outputs the next hidden feature by applying weighted summation on the values.

$$z_i = \sum_{j=1}^{N} \hat{a}_{ij} v_j. \tag{3}$$

z is later fed into a feed forward neural network with a residual connection (He et al., 2016). However, we defer detailed explanations since it is out of the scope of our paper. In practice, self-attention module with multi-head is adopted.

3.2 GRAPH WITH TRANSFORMER

To encode graph structure in Transformer, previous methods focus on encoding graph information into either the attention map or input features fed to Transformer. Graphormer (Ying et al., 2021) adopted two additional terms on the self-attention module to encode graph information on the attention map.

$$a_{ij}^{\text{Graphormer}} = \frac{q_i \cdot k_j}{\sqrt{d_z}} + b_{\psi(i,j)} + \mathcal{E}_{e_{ij}} \cdot w. \tag{4}$$

ψ represents spatial relation between n_i and n_j, which outputs the shortest path distance between the two nodes. Learnable scalar bias $b_{\psi(i,j)}$ encodes spatial relation between two nodes, *e.g.*, b_l is a bias representing two nodes that are l-hop apart. An embedding vector $\mathcal{E}_{e_{ij}}$ is a feature representing edge between the node n_i and the node n_j, and w is a learnable vector. $\mathcal{E}_{e_{ij}} \cdot w$ encodes edge between the two nodes. Moreover, Graphormer adds centrality encoding into the input x which represents the number of edges of a node. Graphormer encodes graph on the attention map with the bias terms parameterized by either spatial relation or edge without considering node feature. However, we additionally consider node-spatial relation and node-edge relation. We will explain the details in Section 4.2.

Dwivedi & Bresson (2020) and Kreuzer et al. (2021) utilize graph Laplacian (Belkin & Niyogi, 2003) $\lambda \in \mathbb{R}^{|N| \times d_x}$ as positional encodings on the input feature x; λ are the top-d_x smallest eigenvectors of $I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ where I is an identity matrix, A is an adjacency matrix and D is a degree matrix. Each cell of a graph Laplacian vector represents partitions after graph min-cut, and neighbouring nodes sharing the many partitions would have similar graph Laplacian. The graph Laplacian λ_i represents the structure of a graph with respect to node n_i.

$$\hat{x_i} = x_i + \lambda_i. \tag{5}$$

Kreuzer et al. (2021) adopt an additional Transformer model f to produce learnable positional encoding: $\hat{x}_i = x_i + \hat{\lambda}_i$ where $\hat{\lambda} = f(\lambda)$. By adding the graph Laplacian into input x, graph information can be encoded in both the attention map and the hidden representations. Their methods lose relative positional information during the linearization of a graph to absolute positional encodings. However, our method encodes a graph directly on the attention map without linearization, thus relative positional information is encoded without loss.

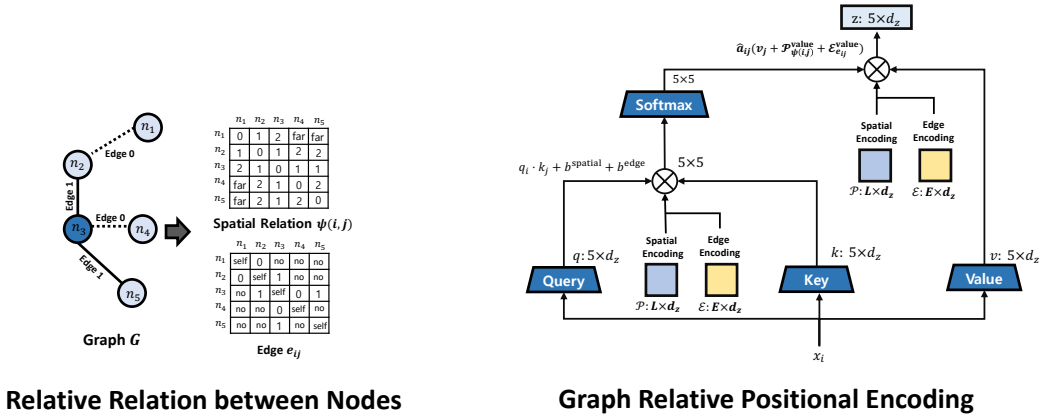

Figure 2: Illustration of the proposed Graph Relative Positional Encoding. Left figure shows an example of how GRPE process relative relation between nodes. In the example we set the L to 2. Right figure describes our self-attention mechanism. Our two relative positional encodings, spatial encoding and edge encoding, are used to encode graph on both attention map and value.

4 Our Approach

We reformulate relative positional encoding for 1D sequence data (Shaw et al., 2018) to incorporate graph-specific properties, such as spatial relation or edges between nodes. Our distinction is twofold. First, we integrate node-spatial and node-edge information on the attention map that was not leveraged in all existing works of graph Transformer. For that, we propose node-aware attention which considers the interactions existing in two pairs: node-spatial relation and node-edge relation. Second, we also encode a graph to the hidden representation of self-attention. For that, we propose graph-encoded values that directly encode relative positional information on the features of value by addition. Our node-aware attention applies the attention mechanism in a node-wise manner, while our graph-encoded value applies the attention mechanism in a channel-wise manner.

4.1 Relative Positional Encoding for Graph

We define two encodings to represent relative positional relation between two nodes in a graph. The first is spatial encoding $\mathcal{P}$, and we define the encodings for query, key, and value respectively: $\mathcal{P}^{\text{query}}, \mathcal{P}^{\text{key}}, \mathcal{P}^{\text{value}} \in \mathbb{R}^{L\times d_z}$. Each vector of $\mathcal{P}$ represents the spatial relation between two nodes, *e.g.*, $\mathcal{P}_l$ represents the spatial relation of two nodes where their shortest path distance is l. L is the maximum shortest path distance that our method considers.

The second is edge encoding $\mathcal{E}$, and we define the encodings for query, key, and value respectively: $\mathcal{E}^{\text{query}}, \mathcal{E}^{\text{key}}, \mathcal{E}^{\text{value}} \in \mathbb{R}^{E\times d_z}$. $\mathcal{E}_{e_{ij}}$ is a vector representing edge between two nodes n_i and n_j. E is the number of types of edge. The spatial encodings and the edge encodings are shared throughout all layers.

4.2 Node-Aware Attention

We propose two terms to encode graph on the attention map with two newly proposed encodings. The first term is b^{spatial}. It encodes graph by considering interaction between node feature and spatial relation in graph.

$$b^{\text{spatial}}_{ij} = q_i \cdot \mathcal{P}^{\text{query}}_{\psi(i,j)} + k_j \cdot \mathcal{P}^{\text{key}}_{\psi(i,j)} \tag{6}$$

The second term is b^{edge}. It encodes graph by considering interaction between node feature and edge in graph.

$$b^{\text{edge}}_{ij} = q_i \cdot \mathcal{E}^{\text{query}}_{e_{ij}} + k_j \cdot \mathcal{E}^{\text{key}}_{e_{ij}} \tag{7}$$

Finally, the two terms are added to scaled dot product attention map to encode graph information.

$$a_{ij} = \frac{q_i \cdot k_j + b_{ij}^{\text{spatial}} + b_{ij}^{\text{edge}}}{\sqrt{d_z}}. \tag{8}$$

Our two terms consider node-spatial relation and node-edge relation, but Graphormer did not consider the interaction with node feature. For instance, any two nodes with the same distance apart have the same bias $b_{\psi(i,j)}$ on Eq. (4), however our b^{spatial} deploys different values according to the node features of query and key.

4.3 Graph-Encoded value

We propose to encode a graph to the hidden features of self-attention, when values are weighted summed with the attention map. We encode both spatial encoding and edge encoding into value via summation:

$$z_i = \sum_{j=1}^{N} \hat{a}_{ij}(v_j + \mathcal{P}_{\psi(i,j)}^{\text{value}} + \mathcal{E}_{e_{ij}}^{\text{value}}). \tag{9}$$

Our method directly encodes graph information into the hidden features of value. The attention weight $\hat{a}$ is applied equally for all channels, while our graph-encoded value enriches the feature of each channel. Therefore, node-aware attention is a node-wise attention mechanism while graph-encoded value is a channel-wise attention mechanism. Our method encodes a graph into the hidden features without losing preciseness of positional information thanks to our relative positional encoding. However, previous approaches encode a graph on the initial input x by linearizing graph *e.g.*, centrality encoding (Ying et al., 2021) or graph Laplacian (Kreuzer et al., 2021; Dwivedi & Bresson, 2020) which loss preciseness of position.

4.4 Complexity Analysis

A naive implementation of computing all pairs of b^{spatial} requires time complexity of $O(N^2 d_z)$, since it requires performing inner product between all node pairs. Instead, we pre-compute an inner product of all possible pairs of node features and spatial encoding vectors $\mathcal{P}$ which requires time complexity of $O(NLd_z)$. Then we assign the pre-computed value according to the indices of node pairs. Likewise, for the b^{edge}, we pre-compute an inner product of all possible pairs of node features and edge encoding vectors $\mathcal{E}^{\text{query}}$ which requires time complexity of $O(NEd_z)$. The time complexity is reduced significantly with our implementation since L and E are much smaller than the number of nodes N.

5 Experiment

5.1 Implementation details

5.1.1 Virtual node

Following Gilmer et al. (2017) and Ying et al. (2021), we adopt a special node called virtual node which is connected to all other nodes. The role of a virtual node is similar to special tokens such as a classification token (Devlin et al., 2018), where its output feature z is used as the input for the branch that predicts downstream tasks. We additionally define two encoding vectors to define both spatial relation $\mathcal{P}_{\text{virtual}}$ and edge $\mathcal{E}_{\text{virtual}}$ for query, key and value respectively. Note that, the virtual node does not involve to find the shortest path between two nodes. Throughout all experiments, we add the virtual node to a graph to perform the downstream tasks.

5.1.2 Spatial Relation

We utilize shortest path distance $\psi(i,j)$ to describe the spatial relation between two nodes n_i and n_j. L is the maximum distance of the shortest path that we consider, and we utilize a special encoding vector $\mathcal{P}_{\text{far}}$ for the node pairs with a distance more than L. For the nodes pairs that are unreachable, we utilize another special encoding vector $\mathcal{P}_{\text{unreachable}}$. Finally, for the pairs that are connected with the virtual node, we utilize another special encoding vector $\mathcal{P}_{\text{virtual}}$.

Table 1: Configurations of models that we utilize throughout our experiments.

Model Configurations	# Params	# Layers	Hidden dim $[d_z]$	FFN layer dim	# Heads
GRPE-Small	489k	12	80	80	8
GRPE-Standard	46.2M	12	768	768	32
GRPE-Large	118.3M	18	1024	1024	32

Table 2: Results on ZINC. * indicates a fine-tuned model. The lower the better.

Method	#Params	MAE
GIN (Xu et al., 2018)	510k	0.526±0.051
GraphSage (Hamilton et al., 2017)	505k	0.398±0.002
GAT (Veličković et al., 2017)	531k	0.384±0.007
GCN (Kipf & Welling, 2016)	505k	0.367±0.011
GatedGCN-PE (Bresson & Laurent, 2017)	505k	0.214±0.006
MPNN* (sum) (Gilmer et al., 2017)	481k	0.145±0.007
PNA (Corso et al., 2020)	387k	0.142±0.010
†GT (Dwivedi & Bresson, 2020)	589k	0.226±0.014
†SAN (Kreuzer et al., 2021)	509k	0.139±0.006
†Graphormer (slim) (Ying et al., 2021)	489k	0.122±0.006
†GRPE-Small (Ours)	489k	**0.094±0.002**

5.1.3 EDGE

Some pair of nodes are not connected with edges. Therefore, we utilize a special encoding vector $\mathcal{E}_{\text{no}}$ for the pairs of node that are not connected with any edges; $\{(n_i, n_j)|i \neq j \text{ and } j \notin \mathcal{N}_i\}_{i=1:|N|}$. For the pair of two identical nodes where $i = j$, we use a special embedding vector $\mathcal{E}_{\text{self}}$. Finally, for the pairs that are connected with the virtual node, we utilize another special encoding vector $\mathcal{E}_{\text{virtual}}$.

5.2 MOLECULE PROPERTY PREDICTION

† indicates the models adopted Transformer for learning graph representation. Bold faced text indicates the best result. We summarize the model configurations of our experiments in Table 1.

We validate our method on the tasks of the molecule property prediction such as OGBG-MolPCBA (MolPCBA) (Hu et al., 2020), OGBG-MolHIV (MolHIV) (Hu et al., 2020) and ZINC (Dwivedi et al., 2020). MolPCBA consists of 437,929 graphs and the task is to predict multiple binary labels indicating various molecule properties. The evaluation metric is average precision (AP). MolHIV is a small dataset that consists of 41,127 graphs. The task is to predict a binary label indicating whether a molecule inhibits HIV virus replication or not. The evaluation metric is area under the curve (AUC). ZINC is also a small dataset that consists of 12,000 graphs, and the task is to regress a molecule property. The evaluation metric is mean absolute error (MAE). All experiments are conducted for 5 times, and we report the mean and the standard deviation of the experiments.

We adopt the linear learning rate decay, and the learning rate starts from $2*10^{-4}$ and ends at $1*10^{-9}$. We set L to 5. For ZINC dataset, we adopt GRPE-Small configuration with less than 500k parameters for a fair comparison. For MolHIV and MolPCBA datasets, we initialize the parameter of the models with the weight of a pretrained model trained on PCQM4M (Hu et al., 2020) dataset.

Table 2 shows the results on ZINC dataset, our model achieve state-of-the-art MAE score. Table 4 shows the results on MolPCBA dataset, our model achieves state-of-the-art AP score. Table 3 shows the results on MolHIV dataset, our model achieves the state-of-the-art AUC with less parameters than Graphormer (Ying et al., 2021).

5.3 OGB LARGE SCALE CHALLENGE

We validate our method on two datasets of OGB large scale challenge (Hu et al., 2020). The two datasets aim to predict the DFT-calculated HOMO-LUMO energy gap of molecules given their molecular graphs. We conduct experiments on both the PCQM4M and PCQM4Mv2 datasets, which are currently the biggest molecule property prediction datasets containing about 4 million graphs

Table 3: Results on MolHIV. * indicates a fine-tuned model. The higher the better.

Method	#Params	AUC (%)
GCN-GraphNorm (Brossard et al., 2020; Cai et al., 2021)	526k	78.83±1.00
PNA (Corso et al., 2020)	326k	79.05± 1.32
PHC-GNN (Le et al., 2021)	111k	79.34± 1.16
DeeperGCN-FLAG (Li et al., 2020)	532k	79.42± 1.20
DGN (Beani et al., 2021)	114k	79.70± 0.97
GIN-VN* (Xu et al., 2018)	3.3M	77.80± 1.82
†Graphormer-FLAG* (Ying et al., 2021)	47.0M	80.51±0.53
†GRPE-Standard* (Ours)	46.2M	**81.39 ± 0.49**

Table 4: Results on MolPCBA. * indicates a fine-tuned model. The higher the better.

Method	#Params	AP (%)
DeeperGCN-VN+FLAG (Li et al., 2020)	5.6M	28.42± 0.43
DGN (Beani et al., 2021)	6.7M	28.85± 0.30
GINE-VN (Brossard et al., 2020)	6.1M	29.17± 0.15
PHC-GNN (Le et al., 2021)	1.7M	29.47± 0.26
GINE-APPNP (Brossard et al., 2020)	6.1M	29.79± 0.30
GIN-VN* (Xu et al., 2018)	3.4M	29.02± 0.17
†Graphormer-FLAG* (Ying et al., 2021)	119.5M	31.39± 0.32
†GRPE-Standard* (Ours)	46.2M	30.77± 0.07
†GRPE-Large* (Ours)	118.3M	**31.50± 0.10**

in total. PCQM4Mv2 contains the DFT-calcuated 3D strcuture of molecules. For our experiments, we only utilize 2D molecular graphs not 3D structures. Throughout experiments, we set L to 5. We adopt a GRPE-Standard for fair comparisons with Graphormer (Ying et al., 2021). We linearly increase learning rate up to $2*10^{-4}$ for 3 epochs and linearly decay learning rate upto $1*10^{-9}$ for 400 epochs. We are unable to measure the test MAE of PCQM4M, because the test dataset is deprecated as PCQM4Mv2 is newly released.

Table 5 shows the results on PCQM4M dataset. Our model achieves the state-of-the-art validation MAE score. Table. 6 shows the results on PCQM4Mv2 dataset. Our model achieves the second best result on both the validation dataset and the test dataset. However, our model use only half of the parameters compared to the best model.

5.4 Ablation study

5.4.1 Components of GRPE

We validate the effect of components of GRPE on ZINC dataset. We adopt GRPE-Small. Table 7 shows the ablation study results. The first row is identical to the plain Transformer without any positional encodings, and it obviously shows the highest error. Adding either b^{spatial} or b^{edge} lowers the error, and using them together further lowers the error. Finally, adding our Graph-Encoded Value does help to improve the performance.

5.4.2 Maximum shortest path distance L

We validate the effects of the maximum shortest path distance L. We adopt GRPE-Standard, and the models are trained from scratch. Figure 3 shows the ablation study result. Increasing L means that a model can identify the position of nodes that are further away. The AUC consistently improves by increasing L from one to four, but L more than four does not further improve performance.

5.4.3 Sharing spatial encoding and edge encoding

We conduct ablation studies about the effects of sharing spatial encoding $\mathcal{P}$ and edge encoding $\mathcal{E}$ for all layers. We adopt GRPE-Small. We conduct five independent runs. The results in Table 8 show that sharing two encodings does improve MAE but not significantly. Empirically, sharing the encodings does not significantly change the performance of a model especially on large datasets.

Table 5: Results on PCQM4M. * indicates the results are from the official leaderboard. VN indicates that the model used virtual node. The lower the better.

Method	#Params	Train MAE	Validate MAE	Test MAE
GCN (Kipf & Welling, 2016)	2.0M	0.1318	0.1691 (0.1684*)	0.1838*
GIN (Xu et al., 2018)	3.8M	0.1203	0.1537 (0.1536*)	0.1678*
GCN-VN (Kipf & Welling, 2016)	4.9M	0.1225	0.1485 (0.1510*)	0.1579*
GIN-VN (Xu et al., 2018)	6.7M	0.1150	0.1395 (0.1396*)	0.1487*
GINE-VN (Brossard et al., 2020)	13.2M	0.1248	0.1430	-
DeeperGCN-VN (Li et al., 2020)	25.5M	0.1059	0.1398	-
†GT (Dwivedi & Bresson, 2020)	0.6M	0.0944	0.1400	-
†GT-wide (Dwivedi & Bresson, 2020)	83.2M	0.0955	0.1408	-
†Graphormer (small) (Ying et al., 2021)	12.5M	0.0778	0.1264	-
†Graphormer (Ying et al., 2021)	47.1M	0.0582	0.1234	**0.1328**
†GRPE-Standard (Ours)	46.2M	**0.0349**	**0.1225**	-

Table 6: Results on PCQM4Mv2. The results of other methods are from the official leaderboard. VN indicates that the model used the virtual node. The lower the better.

Method	#Params	Validate MAE	Test-dev MAE
GCN (Kipf & Welling, 2016)	2.0M	0.1379	0.1398
GIN (Xu et al., 2018)	3.8M	0.1195	0.1218
GCN-VN (Kipf & Welling, 2016)	4.9M	0.1153	0.1152
GIN-VN (Xu et al., 2018)	6.7M	0.1083	0.1084
†EGT (Hussain et al., 2021)	89.3M	**0.0869**	**0.0872**
†GRPE-Standard (Ours)	46.2M	0.0890	0.0898

Table 7: Effects of components of GRPE on ZINC datasets. The lower the better.

b^{spatial}	b^{edge}	Graph-Encoded Value	Test MAE
-	-	-	0.668
✓	-	-	0.267
-	✓	-	0.218
-	-	✓	0.116
✓	✓	-	0.147
✓	✓	✓	**0.093**

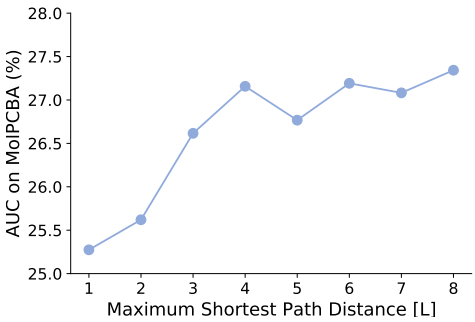

Figure 3: Effects of the maximum shortest path distance L. The higher the better.

Is shared?	#Params	MAE
yes	489k	**0.094**$\pm$ 0.002
no	579k	0.101 $\pm$ 0.003

Table 8: Effects of sharing spatial encoding and edge encoding for all layers. We report MAE on ZINC dataset. The lower the better.

REFERENCES

Dominique Beani, Saro Passaro, Vincent Létourneau, Will Hamilton, Gabriele Corso, and Pietro Liò. Directional graph networks. In *International Conference on Machine Learning*, pp. 748–758. PMLR, 2021.

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

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Rémy Brossard, Oriel Frigo, and David Dehaene. Graph convolutions that can finally model local structure. *arXiv preprint arXiv:2011.15069*, 2020.

Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pp. 1204–1215. PMLR, 2021.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. Edge-augmented graph transformers: Global self-attention is enough for graphs. *arXiv preprint arXiv:2108.03348*, 2021.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Devin Kreuzer, Dominique Beaini, William L Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *arXiv preprint arXiv:2106.03893*, 2021.

Tuan Le, Marco Bertolini, Frank Noé, and Djork-Arné Clevert. Parameterized hypercomplex graph neural networks for graph classification. *arXiv preprint arXiv:2103.16584*, 2021.

Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*, 2020.

Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *arXiv preprint arXiv:2007.02835*, 2020.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? *arXiv preprint arXiv:2106.05234*, 2021.