Graph Inductive Bias in Transformers without Message Passing

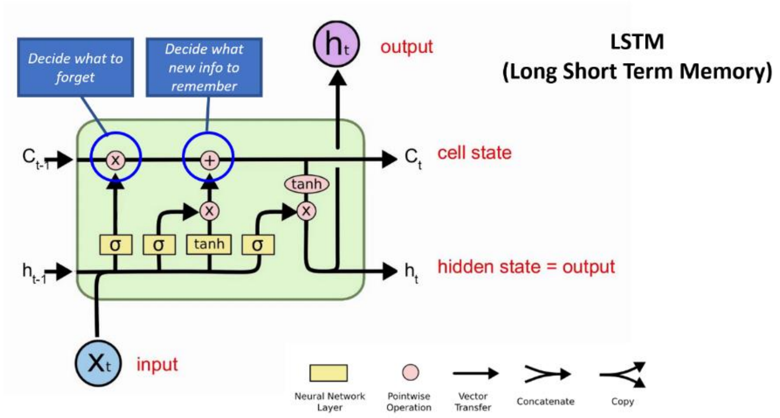
Jiaqing (Steven) Xie

Supervisor: **Florian Grötschla**

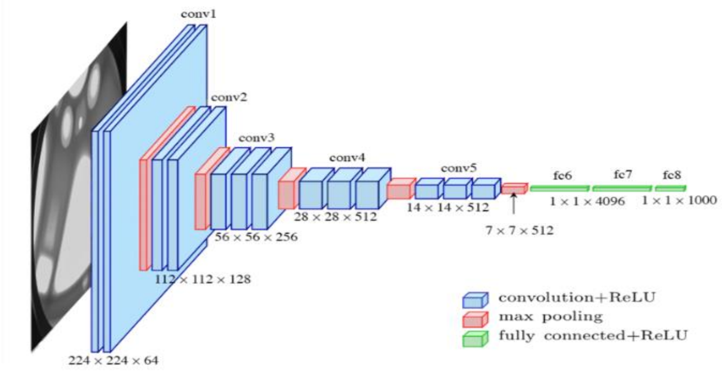
05/03/2024

Individual Modelling

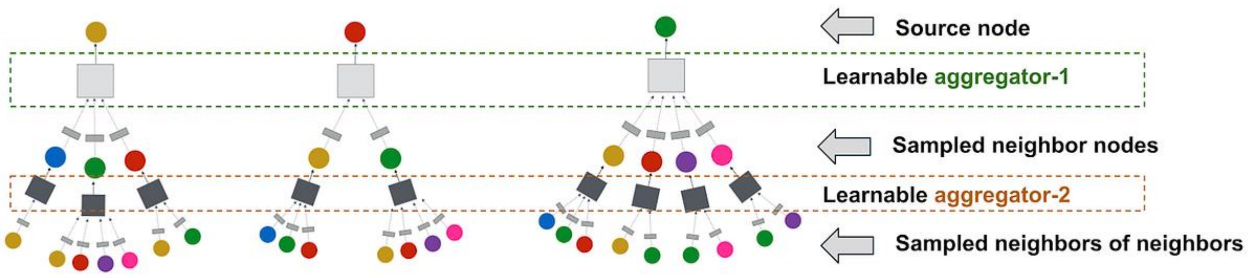
Training LSTM on texts



Training CNN on images



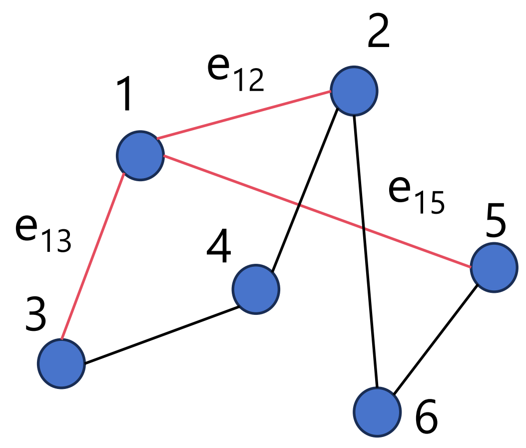
Training **GNN** on **graph**s



Message Passing Neural Networks

$$\mathbf{x}'_i = \mathbf{\Theta}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \cdot h_{\mathbf{\Theta}}(\mathbf{e}_{i,j})$$

Q: Want to update node 1 in **one-hop**



1. Compute messages

   M_{12} = Message(X_1, X_2, e_{12})
   M_{13} = Message(X_1, X_3, e_{13})
   M_{15} = Message(X_1, X_5, e_{15})

   Example: $M_{12} = W * X_1 + X_2 * \mathrm{MLP}(e_{12}) + b$

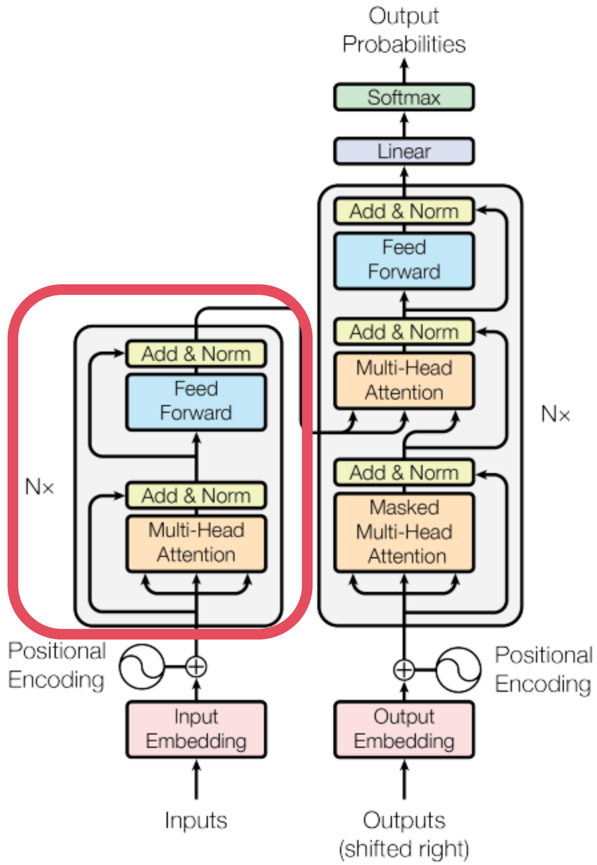
2. Aggregate messages

   Mean aggregation:
   $M_{1\ new} = 1/3 * (M_{12} + M_{13} + M_{15})$

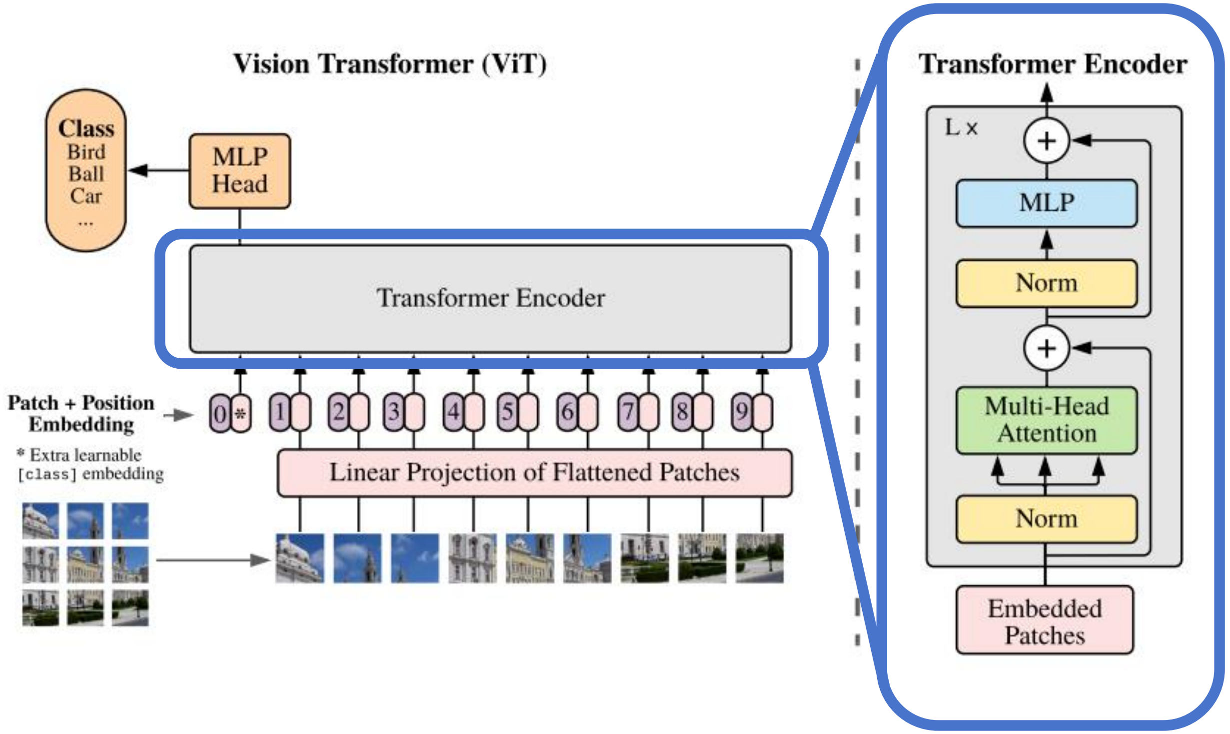
3. Update node feature (for node 1)

   Update by MLP / GRU:
   $X_{1\ new} = W * X_1 + U * M_{1\ new} + b$, where W, U and b are learnable parameters

Unified Encoding Scheme: Transformer



Homogeneous: Transformer Encoder
Heterogeneous: Positional Encoding

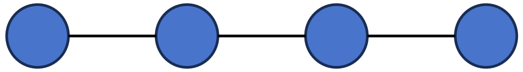


Language Transformer Model (Vaswani 2017 et al.)

Vision Transformer Model (Dosovitskiy 2021 et al.)

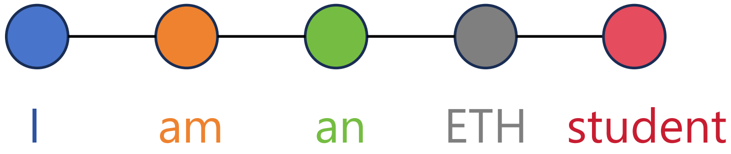
Special Graphs

Path Graph

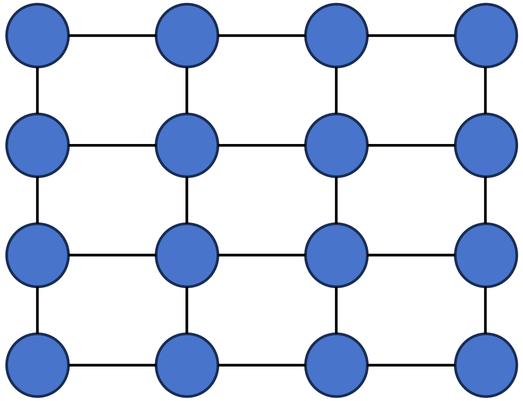


Example:

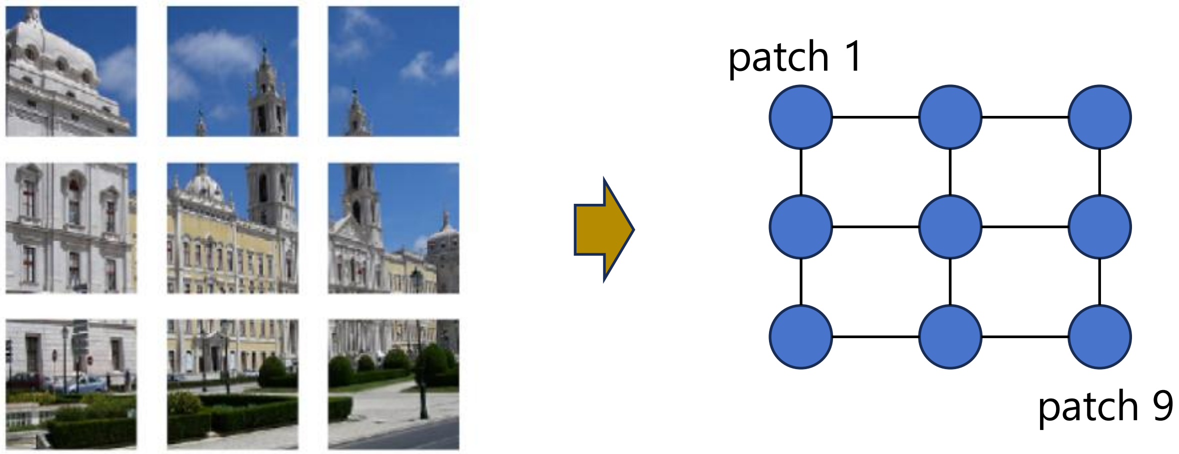
I am an ETH student



Grid (Lattice) Graph



Example:



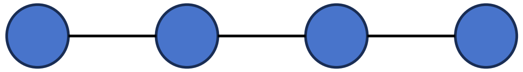
split image into 9 patches

Positional Encoding for Special Graphs

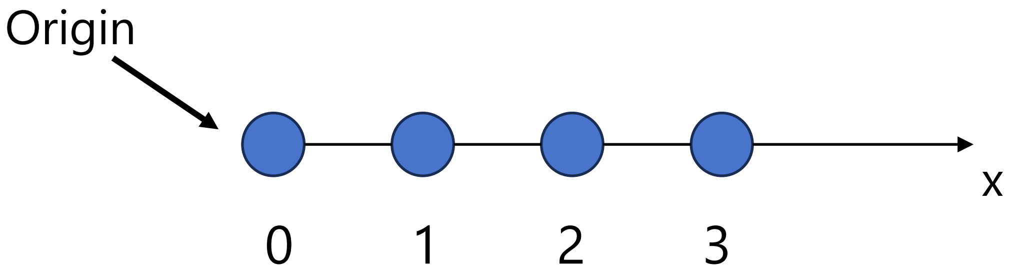
Absolute PE: Sinusoidal

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

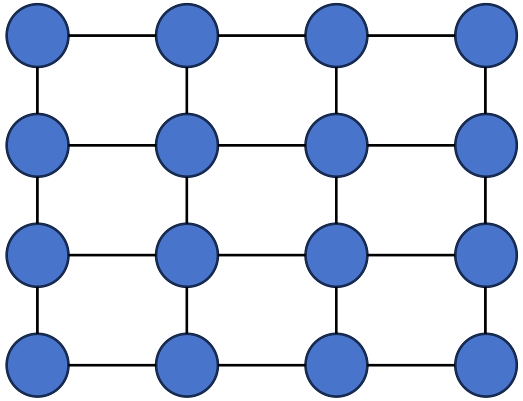
Path Graph



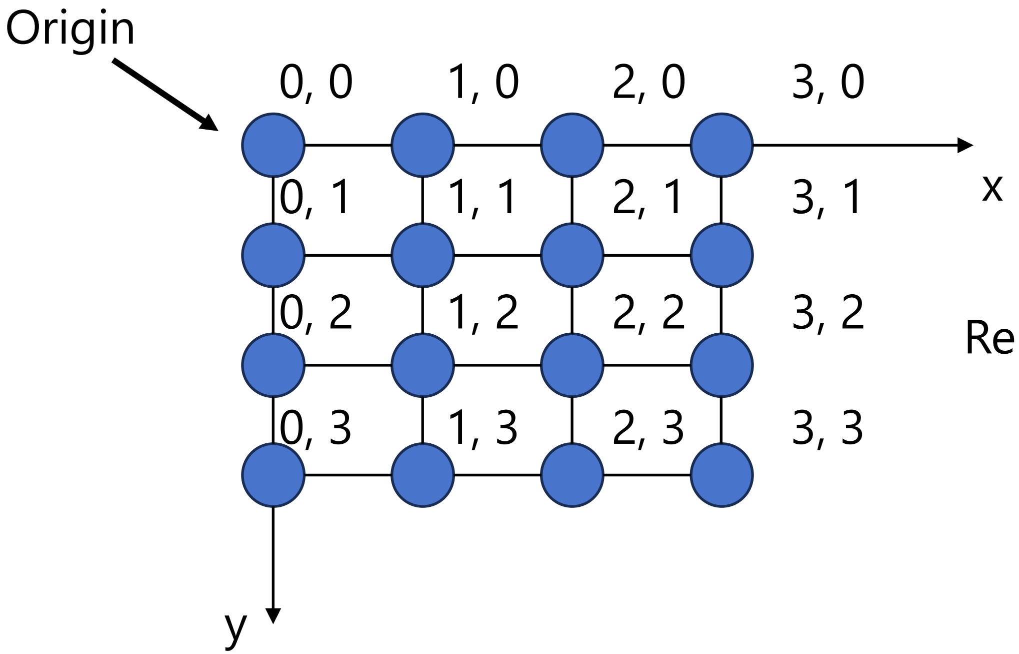
1-dim coordinate system:



Grid (Lattice) Graph



2-dim coordinate system :

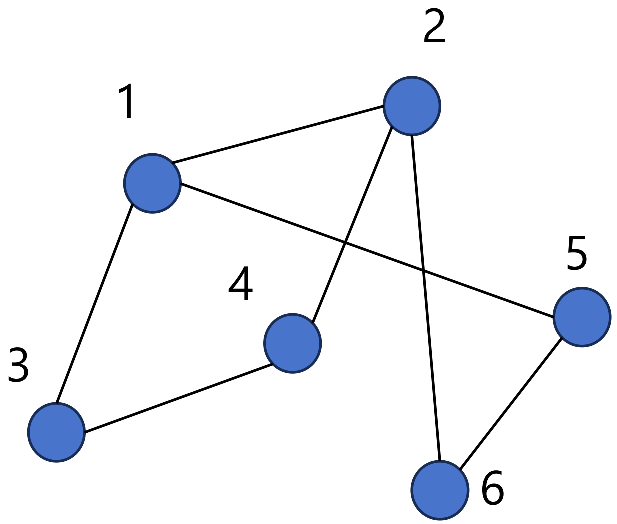


Relative PE:

$PE_{ij} = PE_i - PE_j$

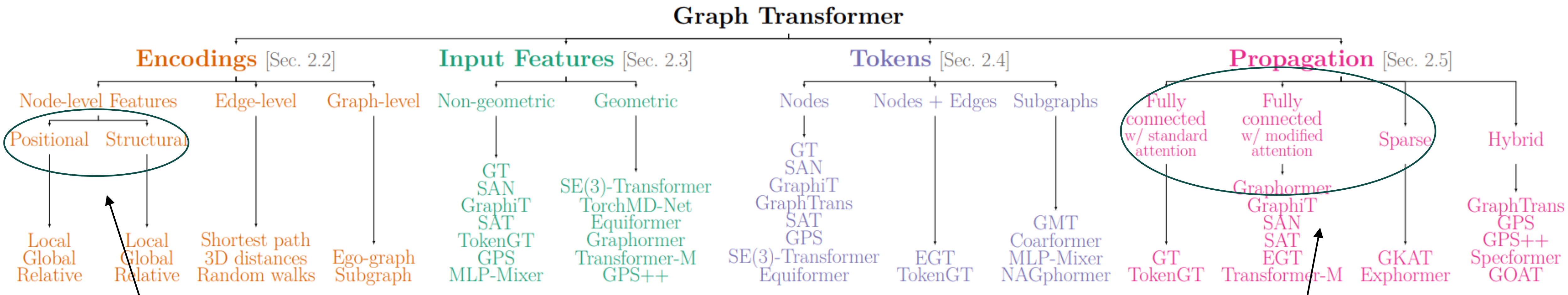
Positional Encoding for General Graphs

It is hard to directly observe positional encodings for general graphs!



- There's no natural Coordinate system for graphs
- Canonical Ordering is limited to planar graphs
- Some solutions: DFS / BFS / Random Walk

Graph Transformers

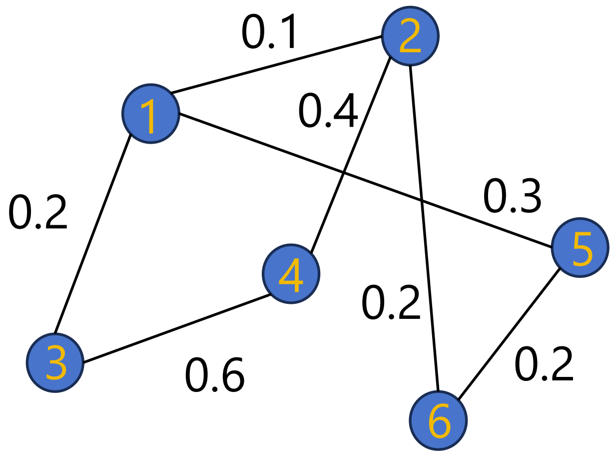


Family of Graph Transformers (Luis et al. 2024)

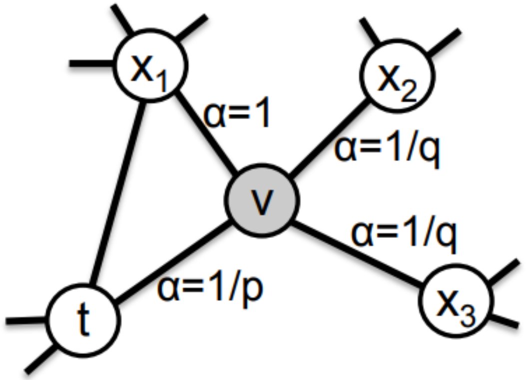
Positional Encoding

Message Passing / Global Attention

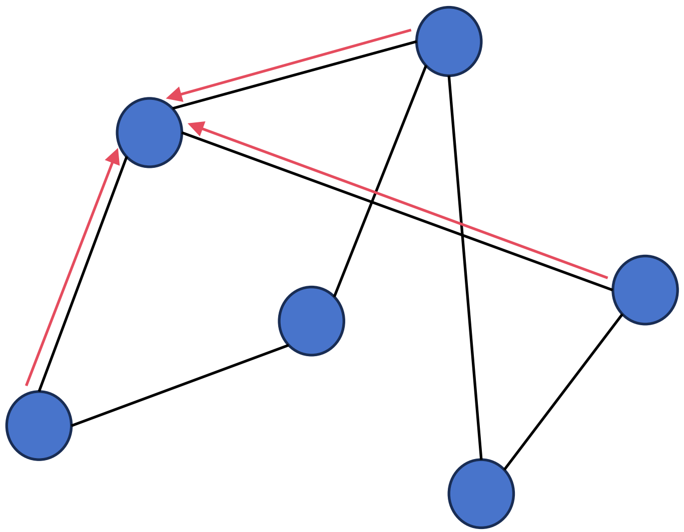
Positional Encoding (PE)

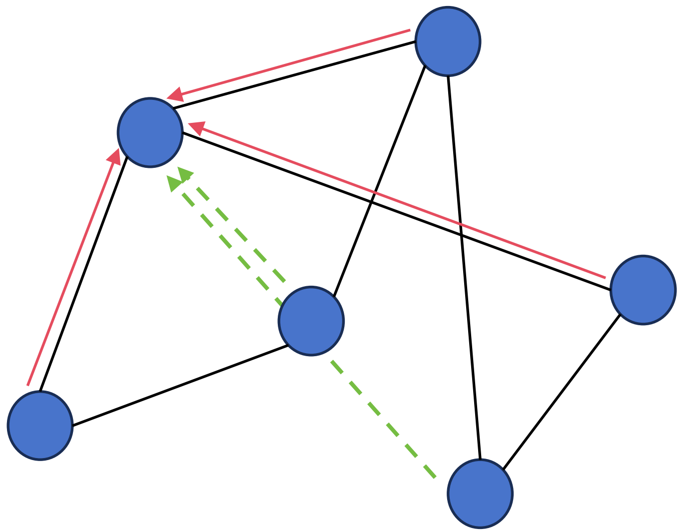


- Shortest Path Distance (SPD)
  - Example: for node 1, SPD PE is [0, 0.1, 0.2, 0.5, 0.3, 0.3]
- Eigenvalue Decomposition on Graph Laplacians
- Random Walk:
  - Example: Walk length = 4, starting from node 1: **(1, 3, 4, 2, 6) => generate a RW corpus**
  - Remember **Word2Vec**
- Node2Vec: Biased Random Walk
  - Explore more: **(1, 3, 4, 2, 6)**
  - Return more: **(1, 3, 1, 2, 1)**



MPNN vs. Global Attention





: Propagation in MPNN

Complexity: O(|V|) for sparse graphs where |V| >> |E|

Capture neighborhood nodes

: Propagation in Global Attention

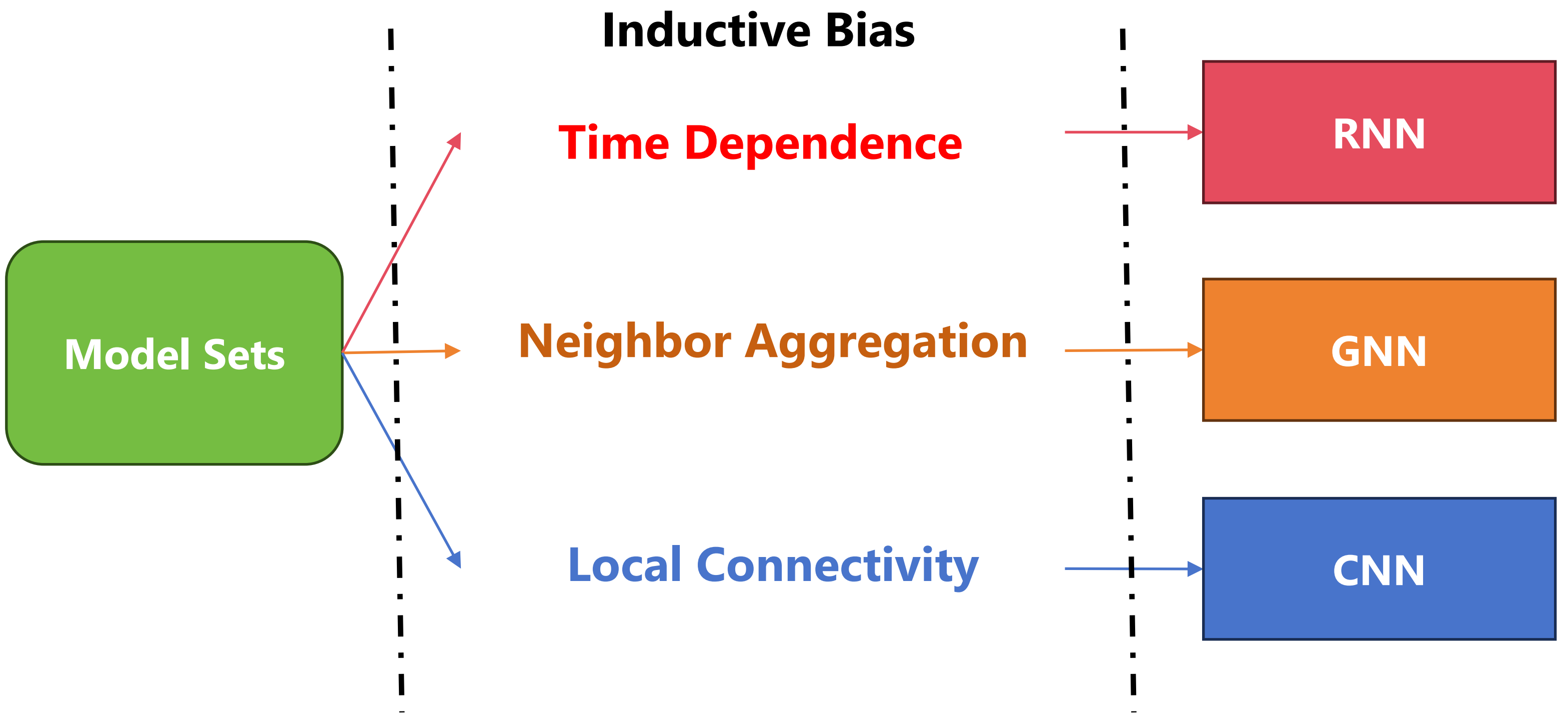
: Additional propagation in graph transformer

Complexity: $O(|V|^2)$

Capture all nodes in graph

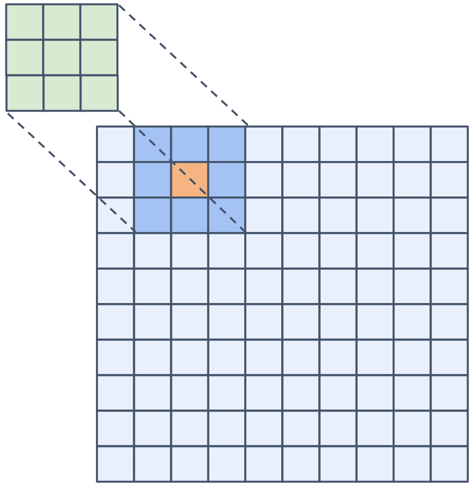
Inductive Bias

Informal definition: model's capability of capturing prior information of data



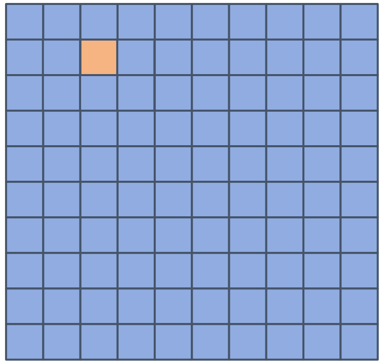
Inductive Bias (CNN vs. Transformer)

CNNs serves locality while self-attention layers are global



Neighbor information is aggregated by the kernel in CNN.

Locality: ☑

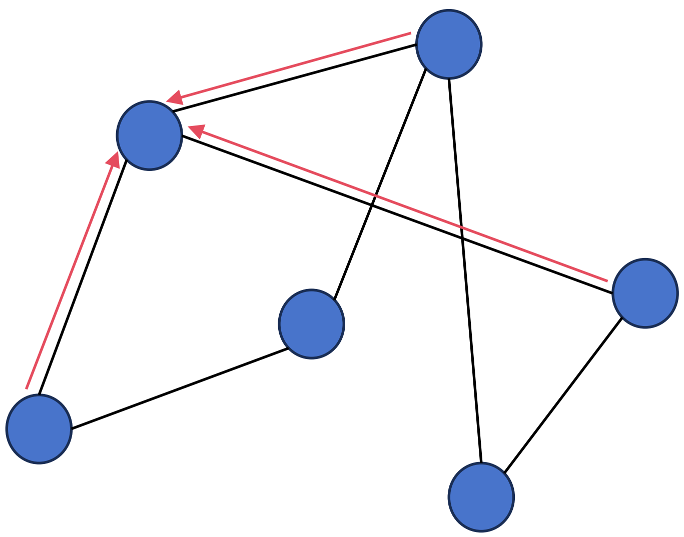


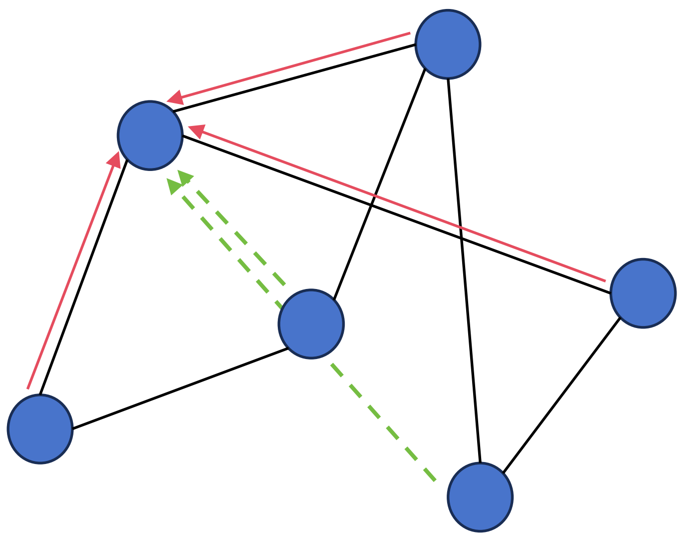
Positions of pixels are unknown for self-attention blocks.

Locality: ☒

Graph Inductive Bias (MPNN vs. Graph Transformer)

MPNNs serves locality while self-attention layers in GT are global





Neighbor nodes' information is aggregated by MPNN

Potential unlinked nodes are supposed to be linked under the settings of graph transformer.

Locality: ☑

Locality: ☒

Pros and cons of MPNN and GT

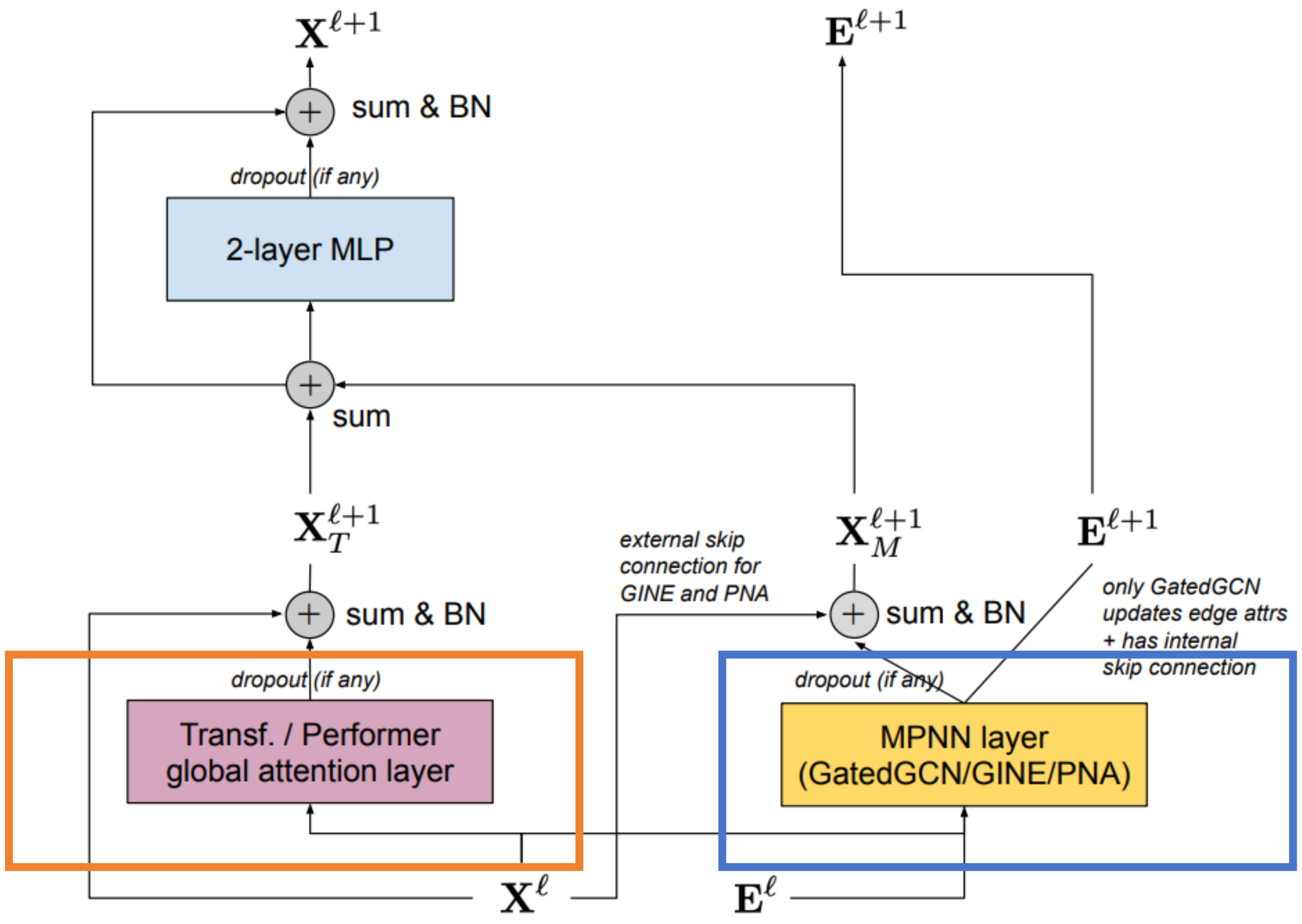
Pros

- MPNN focus on local dependencies. It’s more effective where local graph topology takes matter.
- GT focus on global dependencies. It works well on graph level tasks.

Cons

- MPNN suffers from over-smoothing where all node representations are the same.
- Graph transformers suffer from the missing of graph inductive bias (local topology).

Combine MPNN and GT: GraphGPS



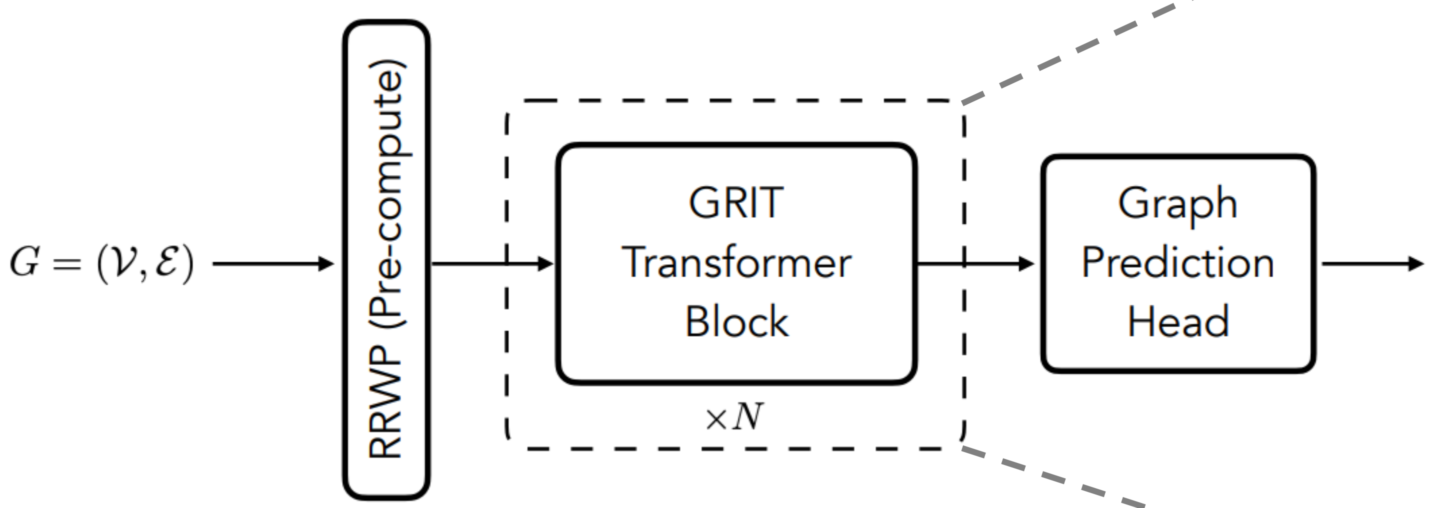
GraphGPS Layer (Rampášek et al. 2022)

Good ☑ : Insert MPNN to GT will bring graph inductive bias in GT.

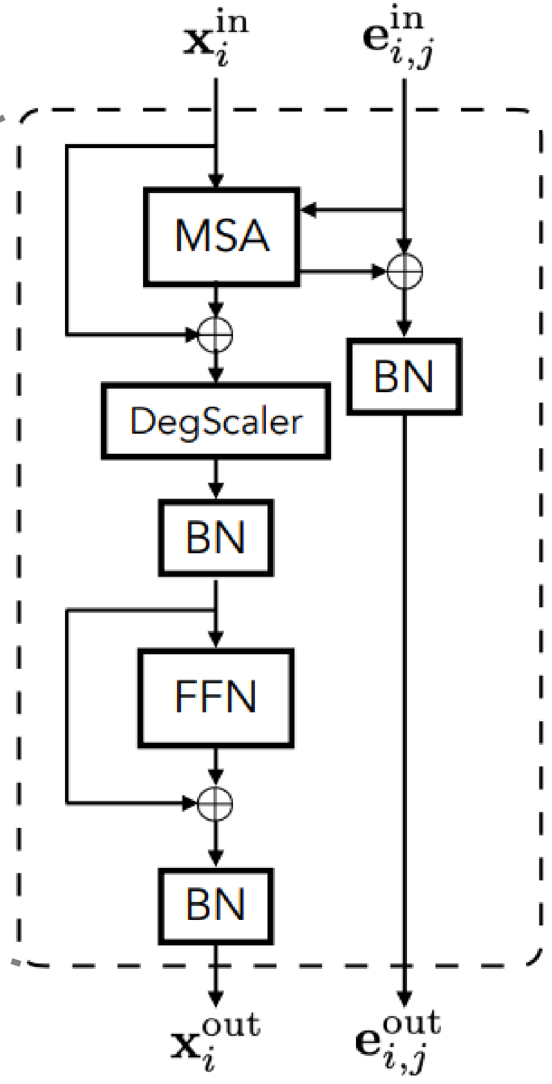
Bad ☒ : New model inherits oversmoothing from MPNN

Without MPNN in GT?
Yes, if PE + GT is as good as MPNN.

Overview: GRIT

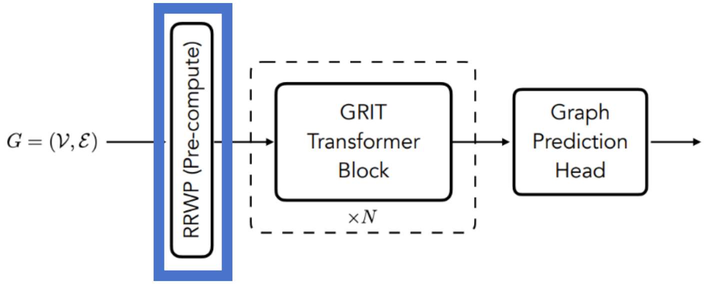


Overall architecture of GRIT
Transformer with Positional Encoding



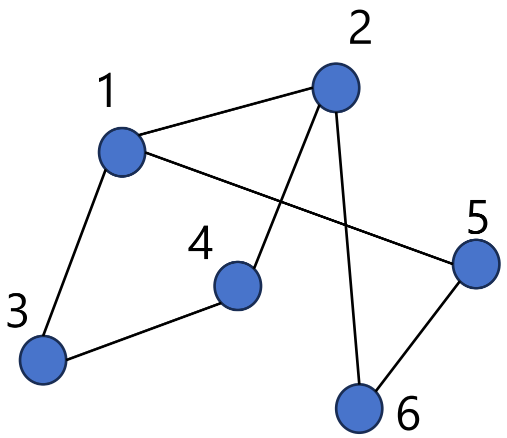
Internal architecture of GRIT

Relative Random Walk Probabilities (RRWP)



- RRWP is a positional encoding method for graph
- Define $\boldsymbol{A}$: Adjacency Matrix
- Define $\boldsymbol{D}$: Diagonal Degree Matrix
- Define $\boldsymbol{M} = \boldsymbol{D}^{-1}\boldsymbol{A}$: prob of node i to node j in one RW

Example:



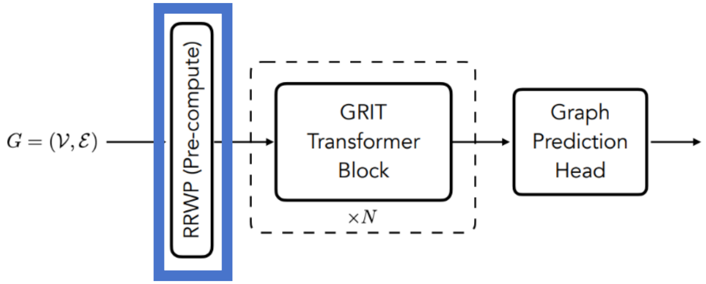
$$\boldsymbol{A} \quad \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{D}^{-1} \quad \begin{bmatrix} 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 \end{bmatrix}$$

$$\boldsymbol{D} \quad \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$\boldsymbol{M} \quad \begin{bmatrix} 0 & 0.333 & 0.333 & 0 & 0.333 & 0 \\ 0.333 & 0 & 0 & 0.333 & 0 & 0.333 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 & 0.5 & 0 \end{bmatrix}$$

Relative Random Walk Probabilities (RRWP)

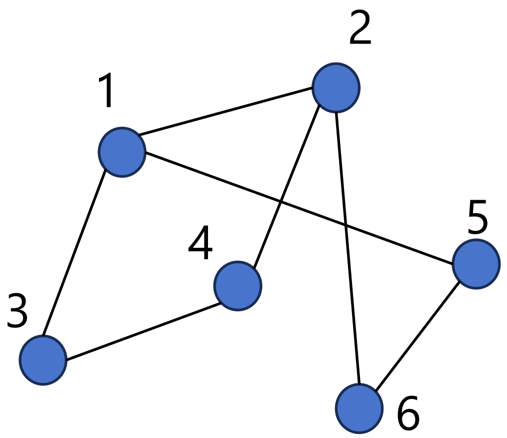


- Define RRWP: $\boldsymbol{P}_{ij} = [\boldsymbol{I}_{ij}, \boldsymbol{M}_{ij}, \boldsymbol{M}_{ij}^2, \ldots, \boldsymbol{M}_{ij}^K]$
- Definition: $\boldsymbol{M}_{ij}^K$: prob of node i to node j by K hops

$$\boldsymbol{I} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{M}^3 \begin{bmatrix} 0.00 & 0.43 & 0.29 & 0.00 & 0.29 & 0.00 \\ 0.43 & 0.00 & 0.00 & 0.29 & 0.00 & 0.29 \\ 0.43 & 0.00 & 0.00 & 0.35 & 0.00 & 0.22 \\ 0.00 & 0.43 & 0.35 & 0.00 & 0.22 & 0.00 \\ 0.43 & 0.00 & 0.00 & 0.22 & 0.00 & 0.35 \\ 0.00 & 0.43 & 0.22 & 0.00 & 0.35 & 0.00 \end{bmatrix}$$

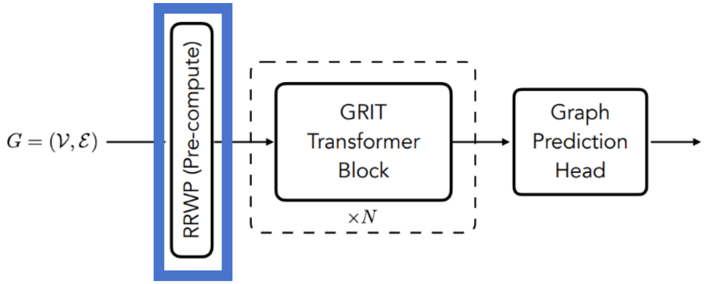
Example:



$$\boldsymbol{M} \begin{bmatrix} 0 & 0.333 & 0.333 & 0 & 0.333 & 0 \\ 0.333 & 0 & 0 & 0.333 & 0 & 0.333 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 & 0.5 & 0 \end{bmatrix}$$

$$\boldsymbol{M}^2 \begin{bmatrix} 0.444 & 0 & 0 & 0.278 & 0 & 0.278 \\ 0 & 0.444 & 0.278 & 0 & 0.278 & 0 \\ 0 & 0.417 & 0.417 & 0 & 0.167 & 0 \\ 0.417 & 0 & 0 & 0.417 & 0 & 0.167 \\ 0 & 0.417 & 0.167 & 0 & 0.417 & 0 \\ 0.417 & 0 & 0 & 0.167 & 0 & 0.417 \end{bmatrix}$$

Relative Random Walk Probabilities (RRWP)

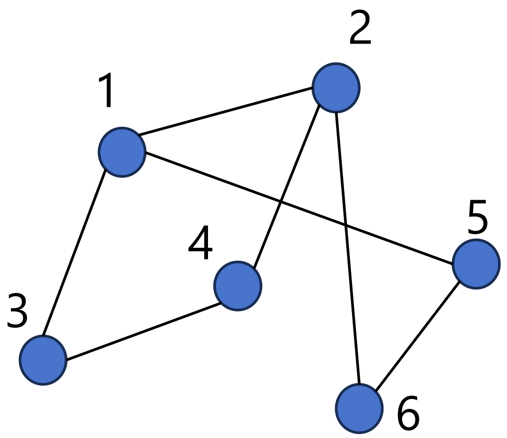


- Define RRWP: $\boldsymbol{P}_{ij} = [\boldsymbol{I}_{ij}, \boldsymbol{M}_{ij}, \boldsymbol{M}_{ij}^{2}, \ldots, \boldsymbol{M}_{ij}^{K}]$
- Definition: $\boldsymbol{M}_{ij}^{K}$: prob of i to j by K hops

$$\boldsymbol{I} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{M}^{3} \begin{bmatrix} 0.00 & 0.43 & 0.29 & 0.00 & 0.29 & 0.00 \\ 0.43 & 0.00 & 0.00 & 0.29 & 0.00 & 0.29 \\ 0.43 & 0.00 & 0.00 & 0.35 & 0.00 & 0.22 \\ 0.00 & 0.43 & 0.35 & 0.00 & 0.22 & 0.00 \\ 0.43 & 0.00 & 0.00 & 0.22 & 0.00 & 0.35 \\ 0.00 & 0.43 & 0.22 & 0.00 & 0.35 & 0.00 \end{bmatrix}$$

Example:



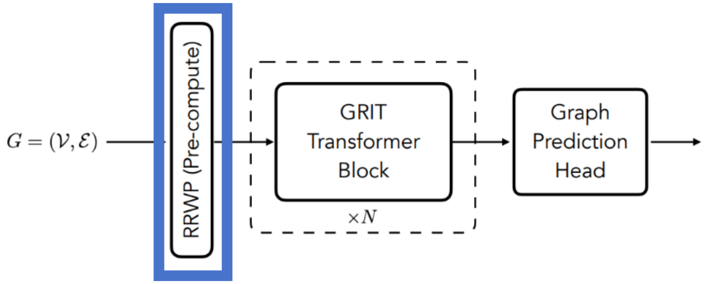
$$\boldsymbol{M} \begin{bmatrix} 0 & 0.333 & 0.333 & 0 & 0.333 & 0 \\ 0.333 & 0 & 0 & 0.333 & 0 & 0.333 \\ 0.5 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 & 0.5 & 0 \end{bmatrix}$$

$$\boldsymbol{M}^{2} \begin{bmatrix} 0.444 & 0 & 0 & 0.278 & 0 & 0.278 \\ 0 & 0.444 & 0.278 & 0 & 0.278 & 0 \\ 0 & 0.417 & 0.417 & 0 & 0.167 & 0 \\ 0.417 & 0 & 0 & 0.417 & 0 & 0.167 \\ 0 & 0.417 & 0.167 & 0 & 0.417 & 0 \\ 0.417 & 0 & 0 & 0.167 & 0 & 0.417 \end{bmatrix}$$

Q: What is $\boldsymbol{P}_{2,3}$ if K = 3?

$$\boldsymbol{P}_{2,3} = [0, 0, 0.278, 0]$$

RRWP + MLP is expressive



Expressive power:

Using RRWP could approximate other PE(s) with MLP

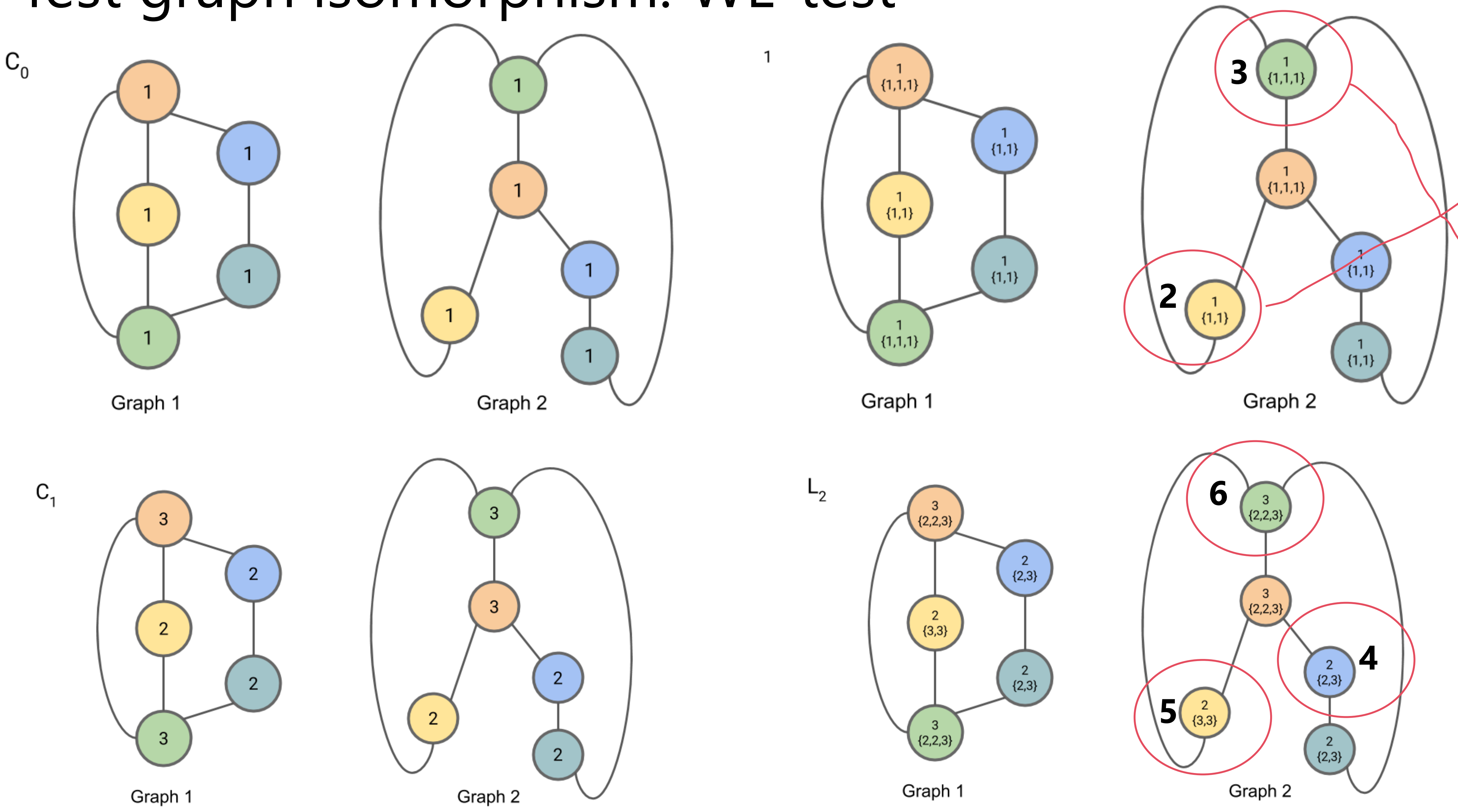
$$(a)\ \ \mathrm{MLP}(\mathbf{P})_{ij} \approx \mathrm{SPD}_{K-1}(i,j)$$

$$(b)\ \ \mathrm{MLP}(\mathbf{P}) \approx \textstyle\sum_{k=0}^{K-1} \theta_k (\mathbf{D}^{-1}\mathbf{A})^k$$

$$(c)\ \ \mathrm{MLP}(\mathbf{P}) \approx \theta_0 \mathbf{I} + \theta_1 \mathbf{A},$$

$$c^{(k+1)}(v) = \mathrm{HASH}\left(c^{(k)}(v), \left\{c^{(k)}(v)\right\}_{u \in N(v)}\right)$$

Test graph isomorphism: WL-test



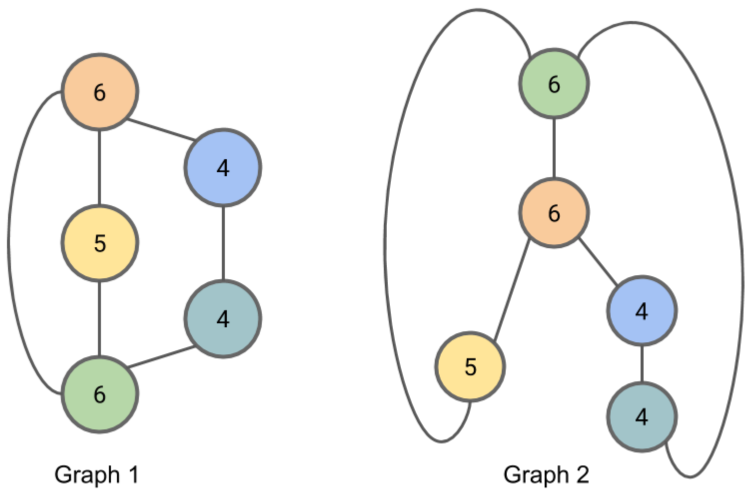
Hash Table

2: {1, {1, 1}}
3: {1, {1, 1, 1}}
4: {2, {2, 3}}
5: {2, {3, 3}}
6: {3, {2, 2, 3}}

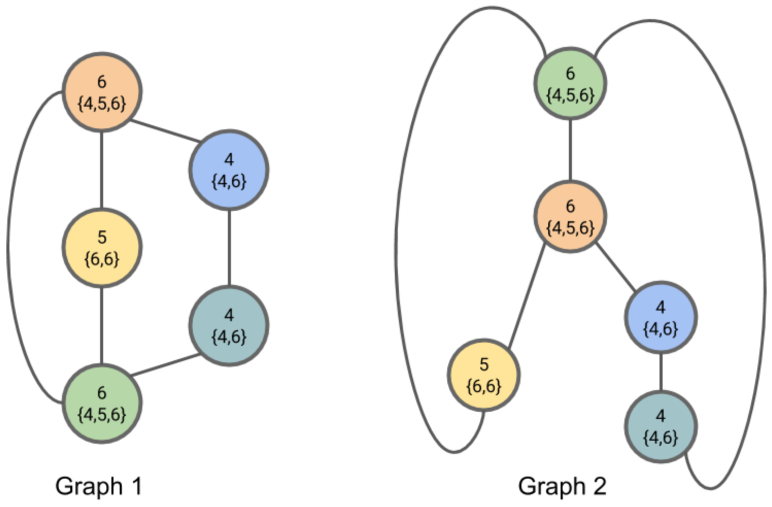
Test graph isomorphism: WL-test

$$c^{(k+1)}(v) = \mathrm{HASH}\left(c^{(k)}(v),\ \left\{c^{(k)}(v)\right\}_{u \in N(v)}\right)$$

C_2



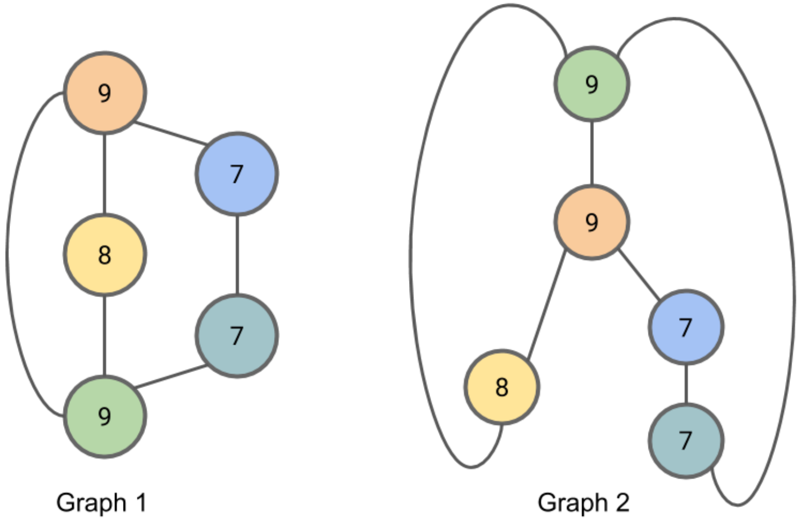
L_3



Hash Table

2: {1, {1, 1}}
3: {1, {1, 1, 1}}
4: {2, {2, 3}}
5: {2, {3, 3}}
6: {3, {2, 2, 3}}
7: {4, {4, 6}}
8: {5, {6, 6}}
9: {6, {4, 5, 6}}

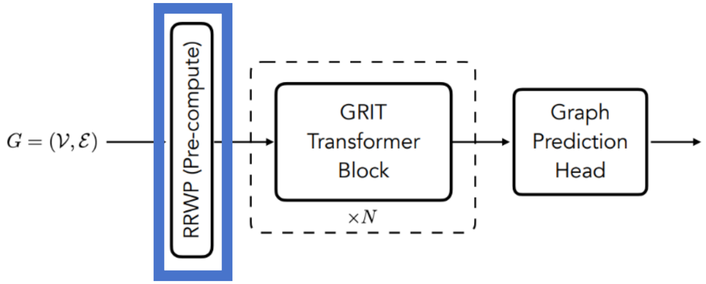
C_3



Careful!

The result is only guaranteed to be correct if the test fails. If the coloring matches, we do not have a guarantee for isomorphism

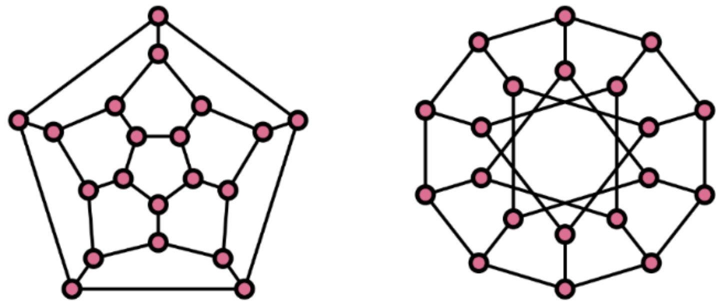
GD-WL: General WL-test with PE



- Intuition: coloring with SPD

$$\chi_G^t(v) = \text{hash}(\{\!\!\{(d_G(v,u), \chi_G^{t-1}(u)) : u \in \mathcal{V}\}\!\!\})$$

- SPD fails with GD-WL (edge weight = 1)



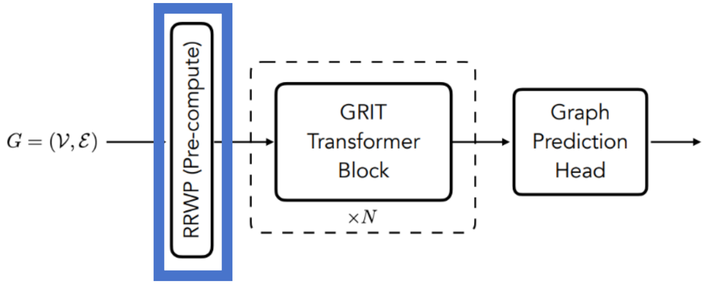
- Reason: for each node, k-hop neighbor array is fixed:

  (3, 6, 6 ,3, 1)

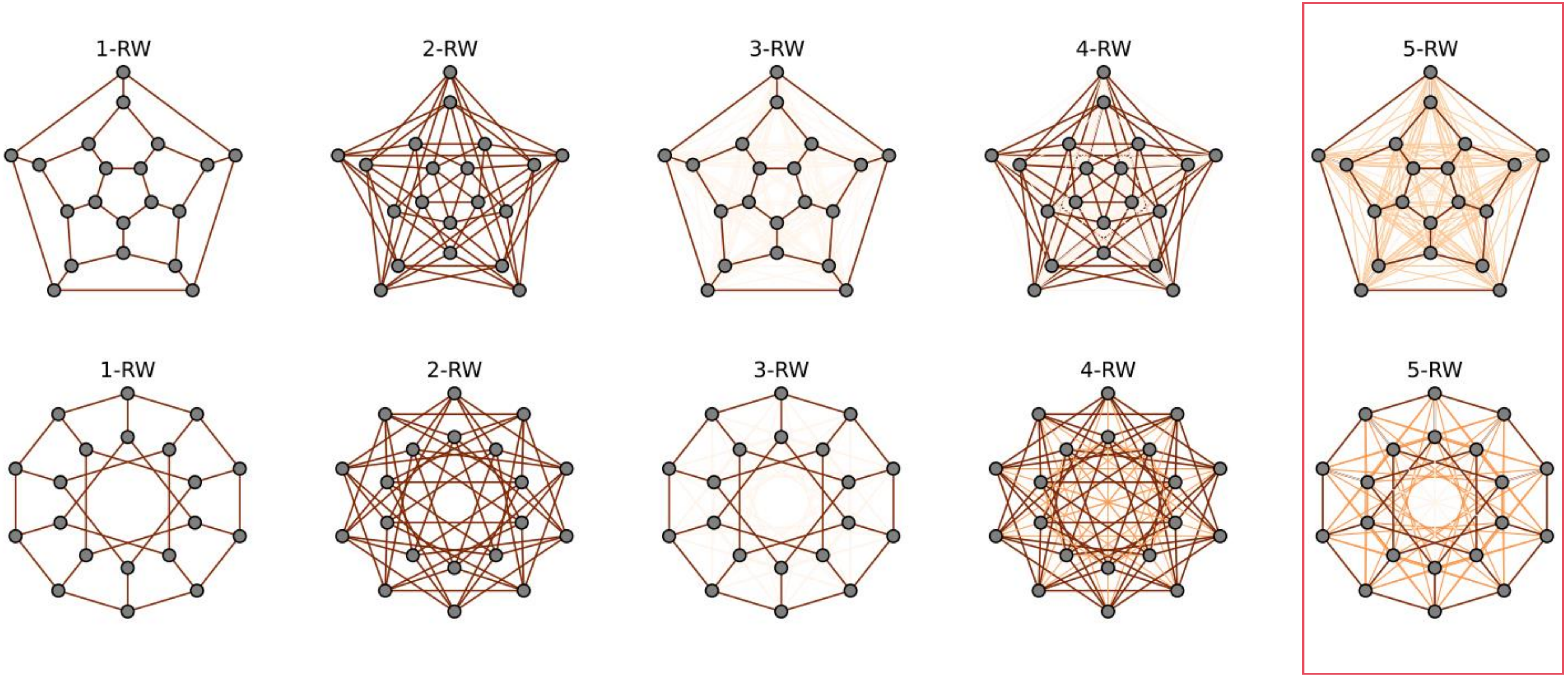
  => {{1, 1, 1}, {2, 2, 2, 2, 2, 2}, {3 ,3 ,3 ,3 ,3 ,3}, {4, 4, 4}, {5}}

Rethinking the expressive power of gnns via graph biconnectivity, Zhang et al., ICLR 2023

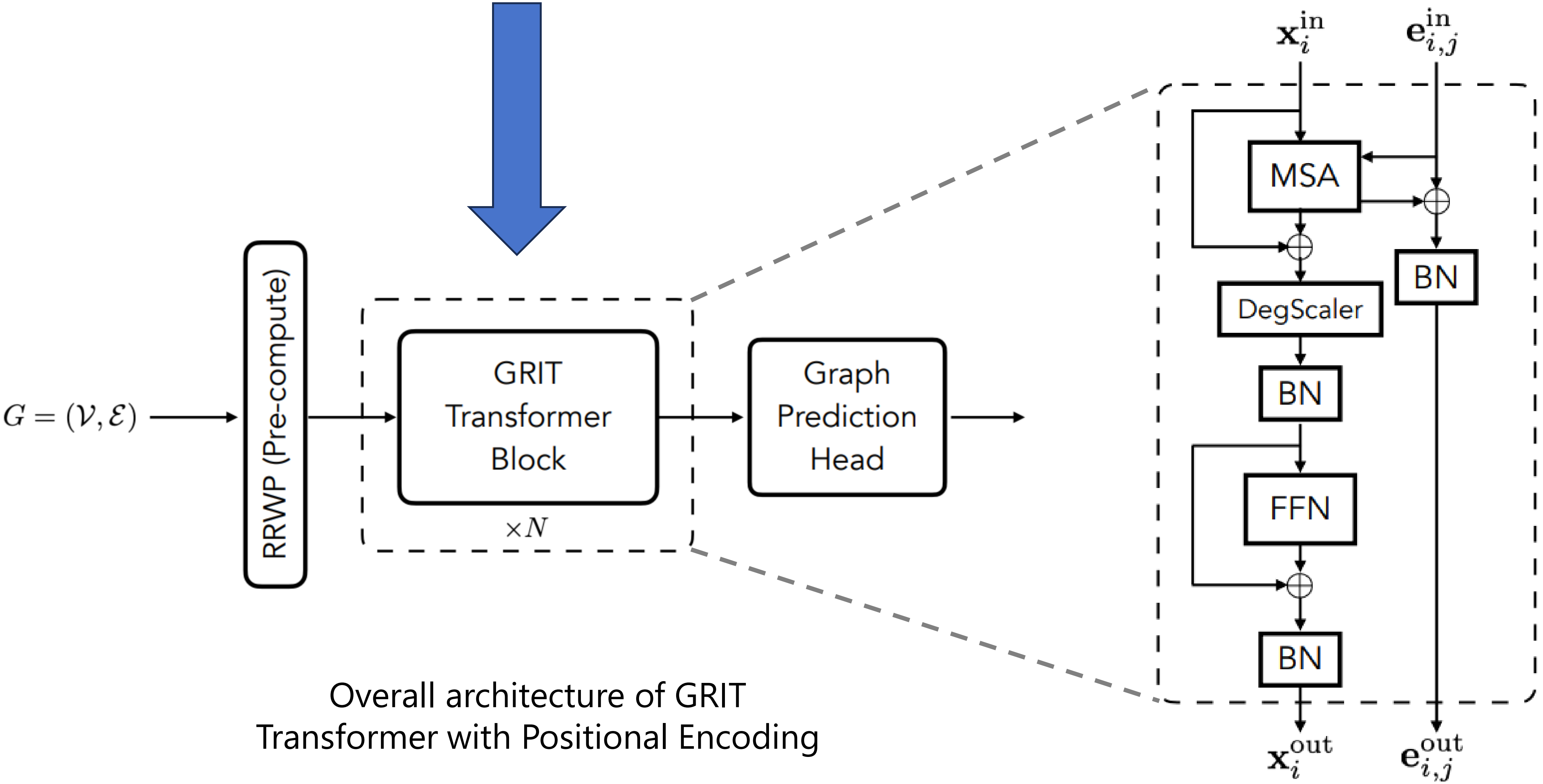
RRWP is more expressive than SPD



RRWP succeeds with GD-WL



Recall: GRIT



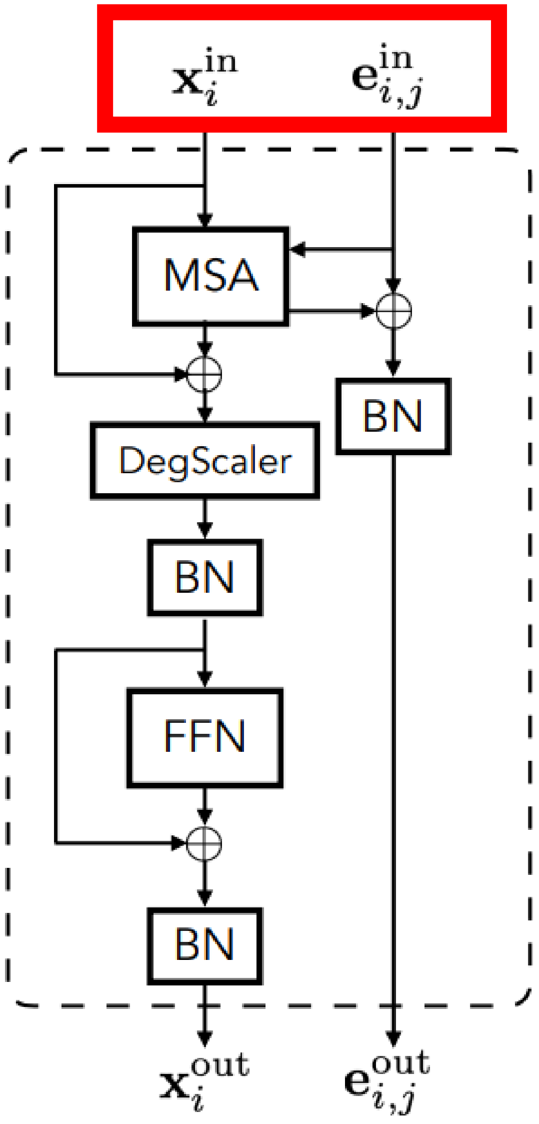
Overall architecture of GRIT
Transformer with Positional Encoding

Internal architecture of GRIT

Flexible Attention

Initialization

$$\mathbf{x}_i = [\mathbf{x}'_i \| \mathbf{P}_{i,i}] \in \mathbb{R}^{d_h+K}$$
$$\mathbf{e}_{i,j} = [\mathbf{e}'_{i,j} \| \mathbf{P}_{i,j}] \in \mathbb{R}^{d_e+K}$$



Flexible Attention

Initialization

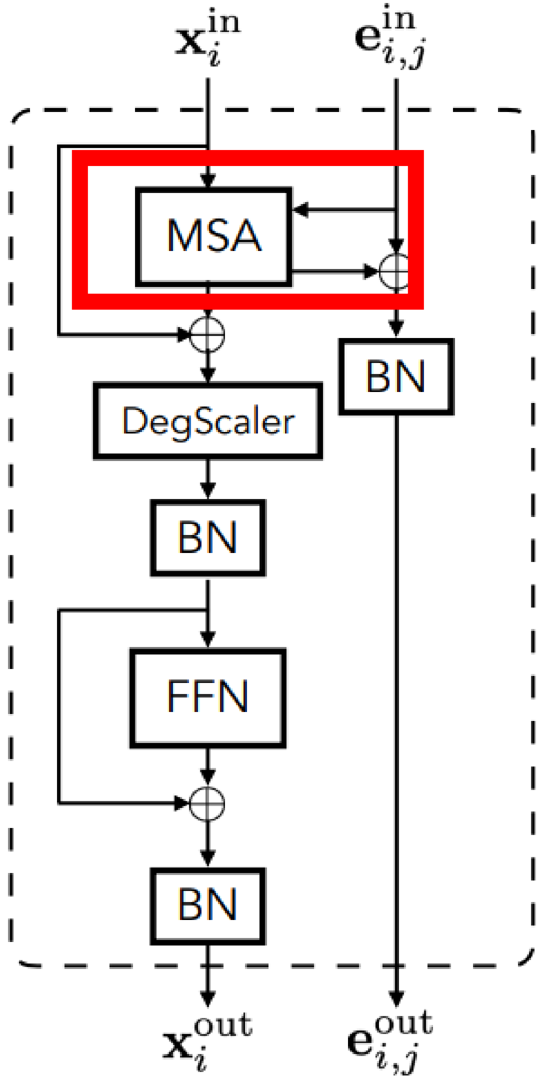
$$\mathbf{x}_i = [\mathbf{x}'_i \| \mathbf{P}_{i,i}] \in \mathbb{R}^{d_h+K}$$
$$\mathbf{e}_{i,j} = [\mathbf{e}'_{i,j} \| \mathbf{P}_{i,j}] \in \mathbb{R}^{d_e+K}$$

Attention Computation：

$$\hat{\mathbf{e}}_{i,j} = \sigma\Big(\rho\left((\mathbf{W}_{\mathrm{Q}}\mathbf{x}_i + \mathbf{W}_{\mathrm{K}}\mathbf{x}_j) \odot \mathbf{W}_{\mathrm{Ew}}\mathbf{e}_{i,j}\right) + \mathbf{W}_{\mathrm{Eb}}\mathbf{e}_{i,j}\Big) \in \mathbb{R}^{d'}$$
$$\alpha_{ij} = \mathrm{Softmax}_{j \in \mathcal{V}}(\mathbf{W}_{\mathrm{A}}\hat{\mathbf{e}}_{i,j}) \in \mathbb{R},$$
$$\hat{\mathbf{x}}_i = \sum_{j \in \mathcal{V}} \alpha_{ij} \cdot (\mathbf{W}_{\mathrm{V}}\mathbf{x}_j + \mathbf{W}_{\mathrm{Ev}}\hat{\mathbf{e}}_{i,j}) \in \mathbb{R}^{d},$$

Recall MPNN:

$$\mathbf{x}'_i = \boldsymbol{\Theta}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j \cdot h_{\boldsymbol{\Theta}}(\mathbf{e}_{i,j})$$



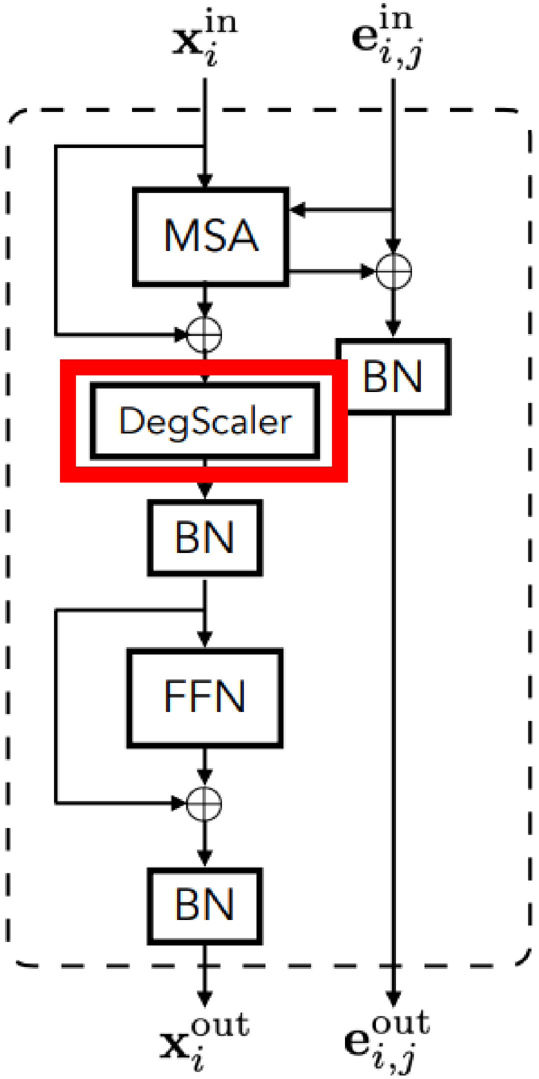
Is it a MPNN? No, we need to compute attention for **each pair of nodes**.

Injecting Degree Information

- Degree information injection:

$$\mathbf{x}_i^{\text{out}'} := \mathbf{x}_i^{\text{out}} \odot \boldsymbol{\theta}_1 + \left(\log(1 + d_i) \cdot \mathbf{x}_i^{\text{out}} \odot \boldsymbol{\theta}_2\right) \in \mathbb{R}^d$$

- Why we need degree scaler?
    - ➢ Attention is innately invariant to node degrees (mean-aggr in GNN)
    Therefore, it reduces expressive power
    - ➢ Adding degree information will introduce inductive bias



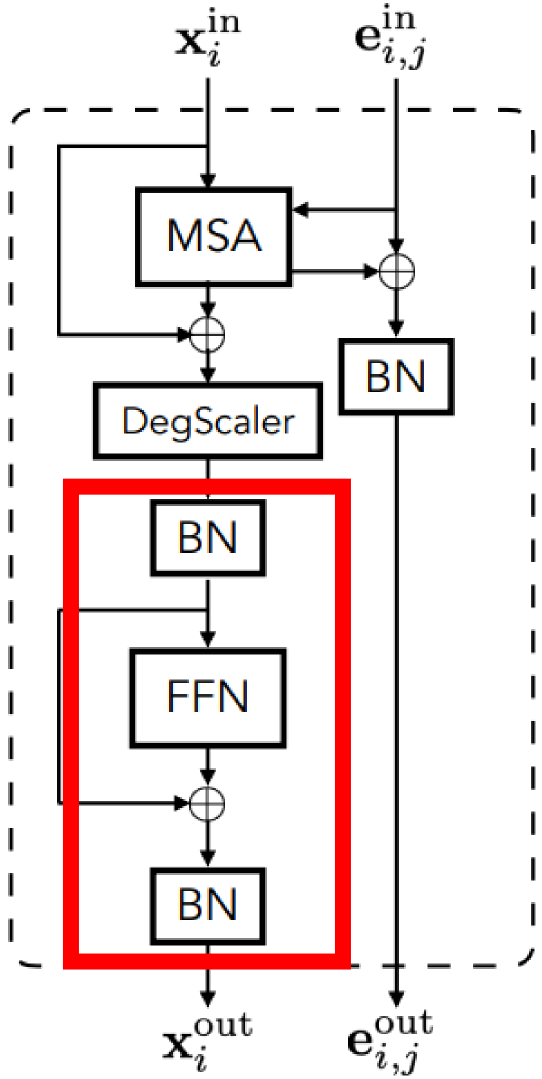
Injecting Degree Information

- Degree information injection:

$$\mathbf{x}_i^{\mathrm{out}'} := \mathbf{x}_i^{\mathrm{out}} \odot \boldsymbol{\theta}_1 + \left(\log(1 + d_i) \cdot \mathbf{x}_i^{\mathrm{out}} \odot \boldsymbol{\theta}_2\right) \in \mathbb{R}^d$$

- Why we need degree scaler?
  - ➢ Attention is innately invariant to node degrees (mean-aggr in GNN) Therefore, it reduces expressive power
  - ➢ Adding degree information will introduce inductive bias

- BatchNorm is favored over LayerNorm
  - ➢ LayerNorm would cancel out the effect brought by degree scaler.



Experiment: Baselines

- SOTA GT: GraphGPS

- Other Graph Transformers:
  - SAN, Graphormer, K-Subgraph SAT, EGT, Graphormer-URPE, Graphormer-GD

- SOTA GNN:
  - CIN, CRaW1, GIN-AK+

- Other GNNs:
  - GIN, GAT, GatedGCN, GatedGCN-LSPE, PNA, DGN, GSN

Experiment: Overview of Benchmarks

Task type:

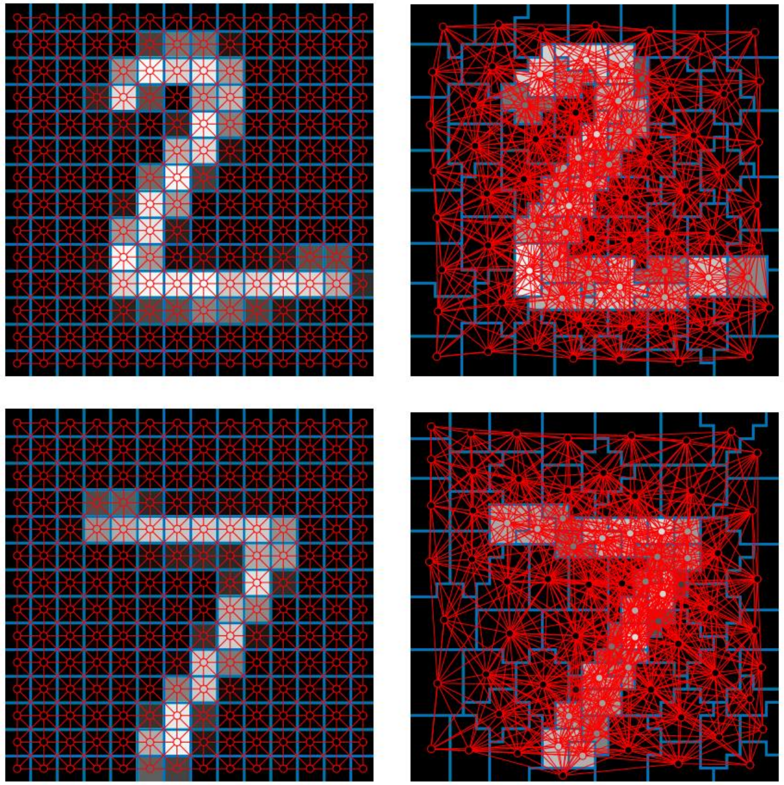
- PATTERN, CLUSTER: node classification (inductive)
- Others: graph classification / graph regression

	Dataset	# Graphs	Avg. # nodes	Avg. # edges
Common Benchmark	ZINC(-full)	12,000 (250,000)	23.2	24.9
	MNIST	70,000	70.6	564.5
	CIFAR10	60,000	117.6	941.1
	PATTERN	14,000	118.9	3,039.3
	CLUSTER	12,000	117.2	2,150.9
Long Range Benchmark	Peptides-func	15,535	150.9	307.3
	Peptides-struct	15,535	150.9	307.3
Open Graph Benchmark	PCQM4Mv2	3,746,620	14.1	14.6

Benchmark 1: Common Benchmarks for GT(s)

- ZINC: molecule dataset
- MNIST, CIFAR10: image classification datasets
- PATTEN, CLUSTER: synthetic datasets sampled from Stochastic Block Model

MNIST superpixels dataset from the "Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs" paper, containing 70,000 graphs with 75 nodes each. Every graph is labeled by one of 10 classes.



Regular grid Superpixels

Benchmark 1: Common Benchmarks for GT(s)

- GRIT has on average better or on par performance when it is compared with GraphGPS
- GRIT has overwhelming advantages when it is compared with GNNs

Model	ZINC	MNIST	CIFAR10	PATTERN	CLUSTER
	MAE↓	Accuracy↑	Accuracy↑	Accuracy↑	Accuracy↑
GCN	0.367 ± 0.011	90.705 ± 0.218	55.710 ± 0.381	71.892 ± 0.334	68.498 ± 0.976
GIN	0.526 ± 0.051	96.485 ± 0.252	55.255 ± 1.527	85.387 ± 0.136	64.716 ± 1.553
GAT	0.384 ± 0.007	95.535 ± 0.205	64.223 ± 0.455	78.271 ± 0.186	70.587 ± 0.447
GatedGCN	0.282 ± 0.015	97.340 ± 0.143	67.312 ± 0.311	85.568 ± 0.088	73.840 ± 0.326
GatedGCN-LSPE	0.090 ± 0.001	–	–	–	–
PNA	0.188 ± 0.004	97.94 ± 0.12	70.35 ± 0.63	–	–
DGN	0.168 ± 0.003	–	72.838 ± 0.417	86.680 ± 0.034	–
GSN	0.101 ± 0.010	–	–	–	–
CIN	0.079 ± 0.006	–	–	–	–
CRaW1	0.085 ± 0.004	97.944 ± 0.050	69.013 ± 0.259	–	–
GIN-AK+	0.080 ± 0.001	–	72.19 ± 0.13	86.850 ± 0.057	–
SAN	0.139 ± 0.006	–	–	86.581 ± 0.037	76.691 ± 0.65
Graphormer	0.122 ± 0.006	–	–	–	–
K-Subgraph SAT	0.094 ± 0.008	–	–	86.848 ± 0.037	77.856 ± 0.104
EGT	0.108 ± 0.009	98.173 ± 0.087	68.702 ± 0.409	86.821 ± 0.020	79.232 ± 0.348
Graphormer-URPE	0.086 ± 0.007	–	–	–	–
Graphormer-GD	0.081 ± 0.009	–	–	–	–
GPS	0.070 ± 0.004	98.051 ± 0.126	72.298 ± 0.356	86.685 ± 0.059	78.016 ± 0.180
GRIT (ours)	$0.059 \pm 0.002^{*}$	98.108 ± 0.111	$76.468 \pm 0.881^{*}$	$87.196 \pm 0.076^{*}$	$80.026 \pm 0.277^{*}$

Benchmark 2: Long Range Graph Benchmark

- Peptides: Amino acid datasets
- Peptides-func: 10-task multi-label classification
- Peptides-struct: 11-task regression
- Long range dataset => **Transformer** captures long range information => **GTs** are better

Model	Peptides-func	Peptides-struct
	AP↑	MAE↓
GCN	0.5930 ± 0.0023	0.3496 ± 0.0013
GINE	0.5498 ± 0.0079	0.3547 ± 0.0045
GatedGCN	0.5864 ± 0.0035	0.3420 ± 0.0013
GatedGCN+RWSE	0.6069 ± 0.0035	0.3357 ± 0.0006
Transformer+LapPE	0.6326 ± 0.0126	$\mathbf{0.2529 \pm 0.0016}$
SAN+LapPE	0.6384 ± 0.0121	0.2683 ± 0.0043
SAN+RWSE	$\mathbf{0.6439 \pm 0.0075}$	0.2545 ± 0.0012
GPS	$\mathbf{0.6535 \pm 0.0041}$	$\mathbf{0.2500 \pm 0.0012}$
GRIT (ours)	$\mathbf{0.6988 \pm 0.0082}^*$	$\mathbf{0.2460 \pm 0.0012}^*$

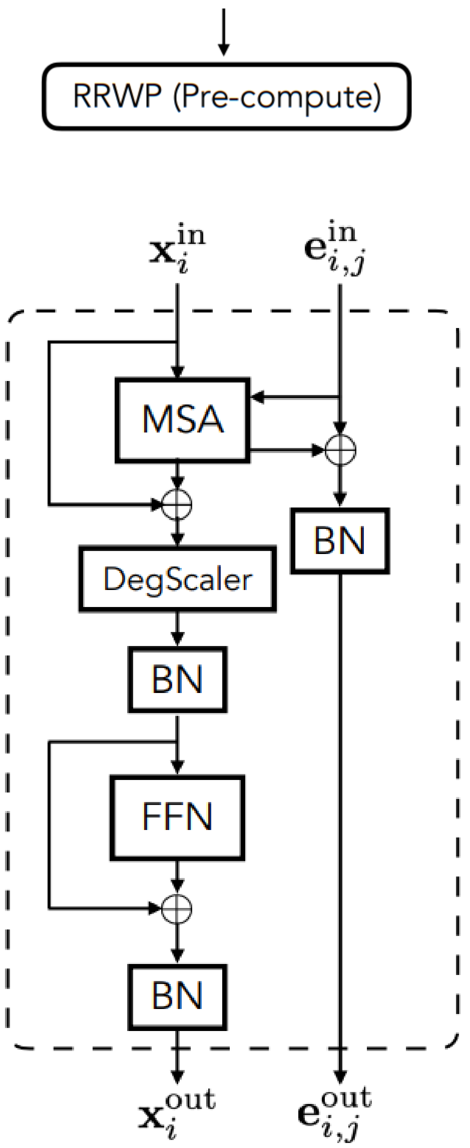
Benchmark 3: Open Graph Benchmark

- PCQM4Mv2 (OGB)
  - Large Scale Graph Datasets (over 3,000,000 graphs)
  - GRIT has on par performance with GraphGPS
  - GRIT has less parameters than GraphGPS

Method	**Model**	**Valid.** (MAE ↓)	**# Param**
MPNNs	GCN	0.1379	2.0M
	GCN-virtual	0.1153	4.9M
	GIN	0.1195	3.8M
	GIN-virtual	0.1083	6.7M
Graph Transformers	GRPE	0.0890	46.2M
	Graphormer	**0.0864**	48.3M
	TokenGT (ORF)	0.0962	48.6M
	TokenGT (Lap)	0.0910	48.5M
	GPS-small	0.0938	6.2M
	GPS-medium	**0.0858**	19.4M
	GRIT (ours)	**0.0859**	16.6M

Ablation Study : Architectural Design Choices

ZINC	MAE ↓
GRIT (ours)	$\mathbf{0.059 \pm 0.002}$
- Remove degree scaler	0.076 ± 0.002
- Remove the update of RRWP	0.066 ± 0.005
- Global-attn. $\rightarrow$ Sparse-attn.	0.066 ± 0.002
- Degree scaler $\rightarrow$ Degree encoding	0.072 ± 0.005
- GRIT-attn. $\rightarrow$ Graphormer-attn.	0.117 ± 0.028
- RRWP $\rightarrow$ RWSE	0.081 ± 0.010
- RRWP $\rightarrow$ SPDPE	0.067 ± 0.002



Opinion: Future works

Advantages:

- Fewer params compared to other GTs
- Importance of positional encodings (GRIT)

Disadvantages:

- Complexity of attention: $O(|V|^2)$
- Upper bound on expressive power

Conclusion

- Design choices for including graph inductive bias in GT (PE / MPNN)
- RRWP encodings are expressive
- RRWP initialization is more expressive than SPD under GD-WL tests
- GRIT is new SOTA graph transformer which excludes message passing

Any Questions

Others: Expressive power of GNN

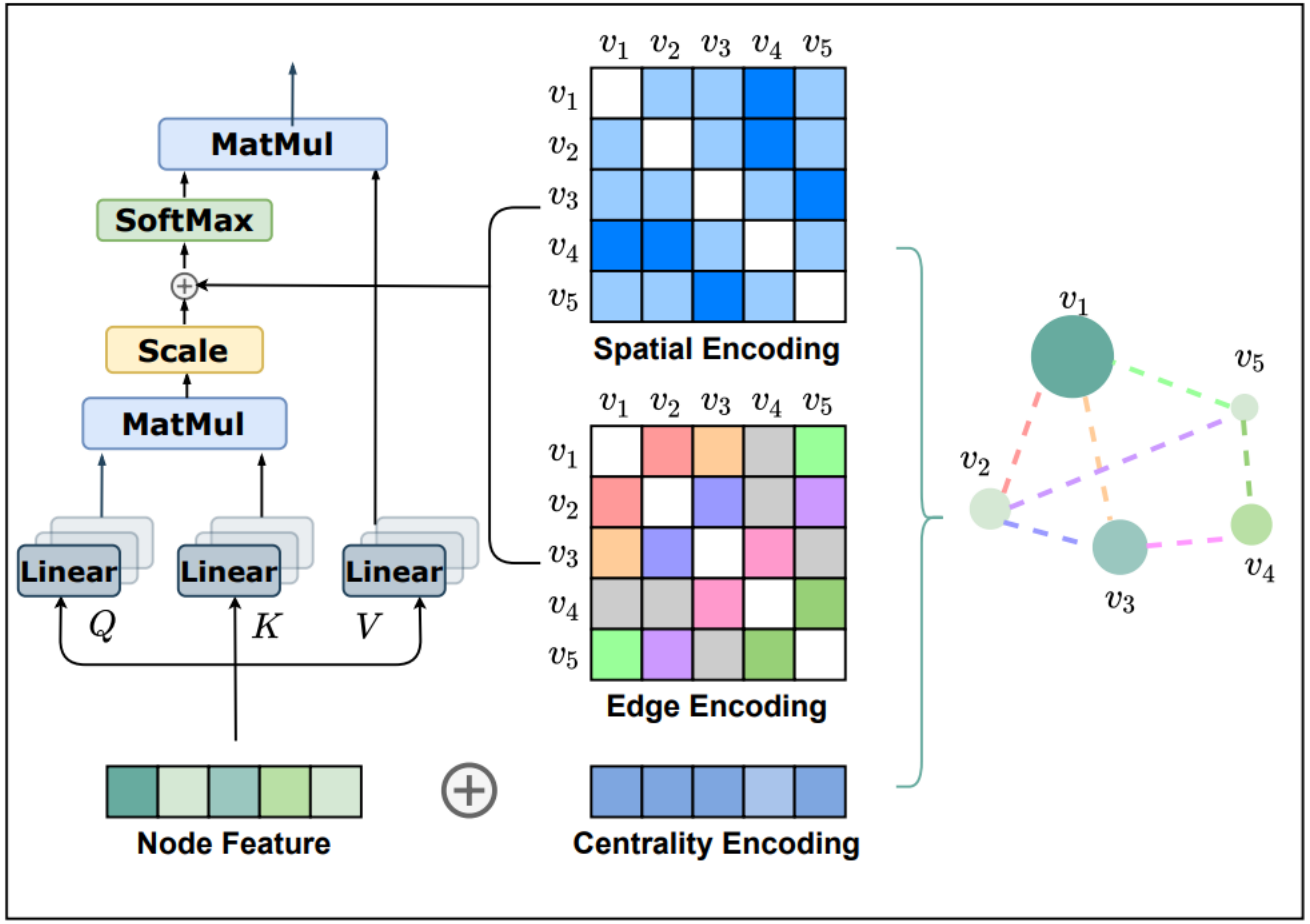
GIN is as powerful as WL test

$$c^{(k+1)}(v) = \boxed{MLP_\theta}\left(\boxed{(1+\epsilon)\cdot MLP_\psi\left(c^{(k)}(v)\right)} + \boxed{\sum_{u \in N(v)} MLP_\psi\left(c^{(k)}(u)\right)}\right)$$

$$c^{(k+1)}(v) = \boxed{\mathbf{HASH}}\left(\boxed{c^{(k)}(v)}, \boxed{\left\{c^{(k)}(v)\right\}_{u \in N(v)}}\right)$$

How Powerful are Graph Neural Networks?, Xu et al., 2018

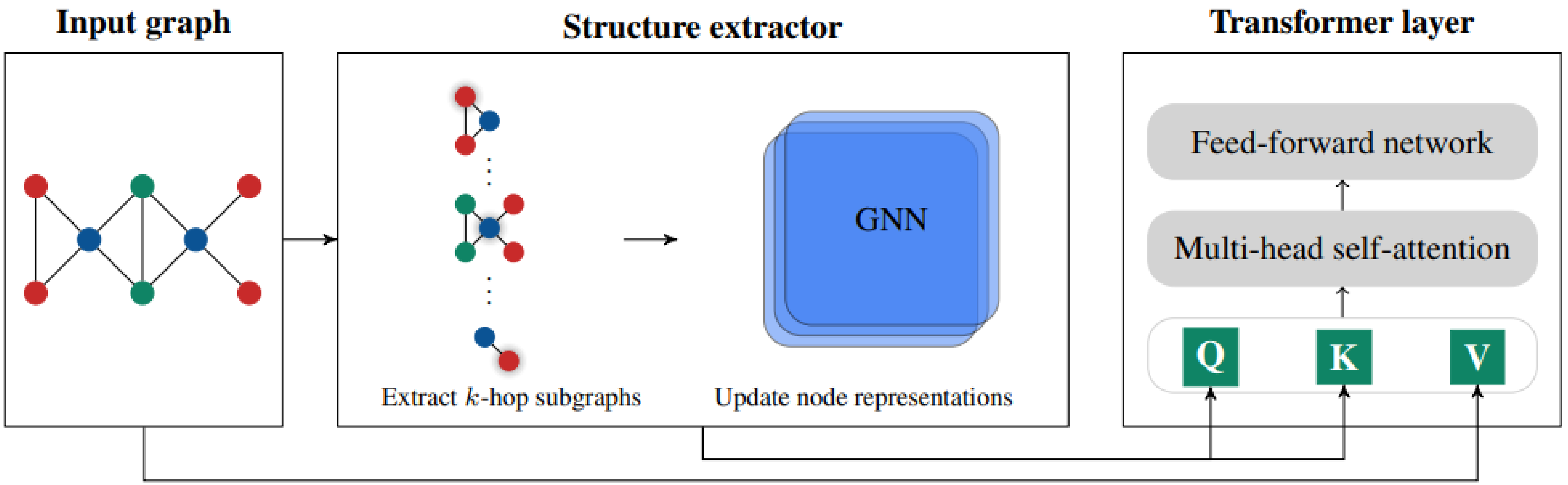
Others: Graphormer Module



Do Transformers Really Perform Bad for Graph Representation?, Ying et al., NeurIPS 2021

Others: Graph Transformer w/ Both Attention

- Structure-Aware



Structure-Aware Transformer for Graph Representation Learning, Chen et al., ICML 2022

Others: Experimental Complexity

Table 12. Runtime and GPU memory for SAN (Kreuzer et al., 2021), GraphGPS (Rampášek et al., 2022) and GRIT (Ours) on ZINC with batch size 32. The timing is conducted on a single NVIDIA V100 GPU and 20 threads of Intel(R) Xeon(R) GOld 6140 CPU @ 2.30GH.

ZINC	SAN	GraphGPS	GRIT (Ours)
MAE $\downarrow$	0.139 ± 0.006	0.070 ± 0.004	0.059 ± 0.002
PE Precompute-time	10 sec	11 sec	11 sec
GPU Memory	2291 MB	1101 MB	1865 MB
Training time	57.9 sec/epoch	24.3 sec/epoch	29.4 sec/epoch

Others: Detailed dataset descriptions

Dataset	# Graphs	Avg. # nodes	Avg. # edges	Directed	Prediction level	Prediction task	Metric
ZINC(-full)	12,000 (250,000)	23.2	24.9	No	graph	regression	Mean Abs. Error
MNIST	70,000	70.6	564.5	Yes	graph	10-class classif.	Accuracy
CIFAR10	60,000	117.6	941.1	Yes	graph	10-class classif.	Accuracy
PATTERN	14,000	118.9	3,039.3	No	inductive node	binary classif.	Weighted Accuracy
CLUSTER	12,000	117.2	2,150.9	No	inductive node	6-class classif.	Accuracy
Peptides-func	15,535	150.9	307.3	No	graph	10-task classif.	Avg. Precision
Peptides-struct	15,535	150.9	307.3	No	graph	11-task regression	Mean Abs. Error
PCQM4Mv2	3,746,620	14.1	14.6	No	graph	regression	Mean Abs. Error

Others: Hyper-parameter settings

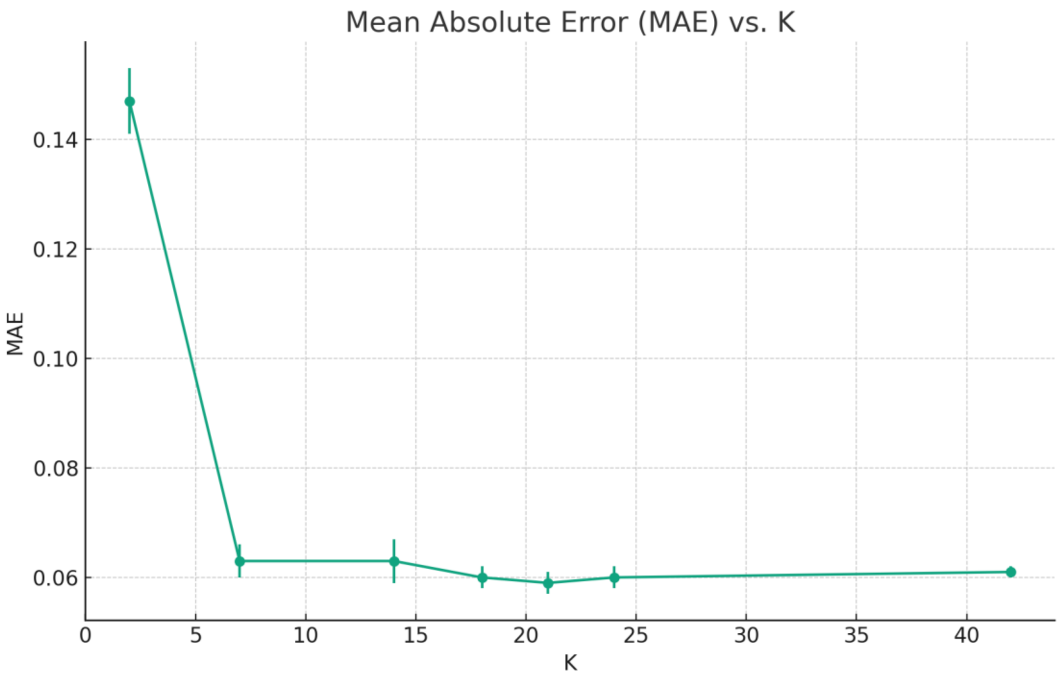
Hyperparameter	ZINC/ZINC-full	MNIST	CIFAR10	PATTERN	CLUSTER
# Transformer Layers	10	3	3	10	16
Hidden dim	64	52	52	64	48
# Heads	8	4	4	8	8
Dropout	0	0	0	0	0.01
Attention dropout	0.2	0.5	0.5	0.2	0.5
Graph pooling	sum	mean	mean	–	–
PE dim (RW-steps)	21	18	18	21	32
PE encoder	linear	linear	linear	linear	linear
Batch size	32/256	16	16	32	16
Learning Rate	0.001	0.001	0.001	0.0005	0.0005
# Epochs	2000	200	200	100	100
# Warmup epochs	50	5	5	5	5
Weight decay	$1e-5$	$1e-5$	$1e-5$	$1e-5$	$1e-5$
# Parameters	473,473	102,138	99486	477,953	432,206

Benchmark 3: Large Dataset

- ZINC-full Dataset
  - 250,000 molecule graphs
  - Higher order GNNs are included in baselines
  - Positional encoding enhanced GNNs are also included

Method	**Model**	**ZINC-full** (MAE $\downarrow$)
MPNNs	GIN	0.088 ± 0.002
	GraphSAGE	0.126 ± 0.003
	GAT	0.111 ± 0.002
	GCN	0.113 ± 0.002
	MoNet	0.090 ± 0.002
Higher-order GNNs	δ-2-GNN	0.042 ± 0.003
	δ-2-LGNN	0.045 ± 0.006
PE-GNN	SignNet	$\mathbf{0.024 \pm 0.003}$
Graph Transformers	Graphormer	0.052 ± 0.005
	Graphormer-URPE	0.028 ± 0.002
	Graphormer-GD	$\mathbf{0.025 \pm 0.004}$
	GRIT (ours)	$\mathbf{0.023 \pm 0.001}$

Ablation Study 2: Parameter K of RRWP

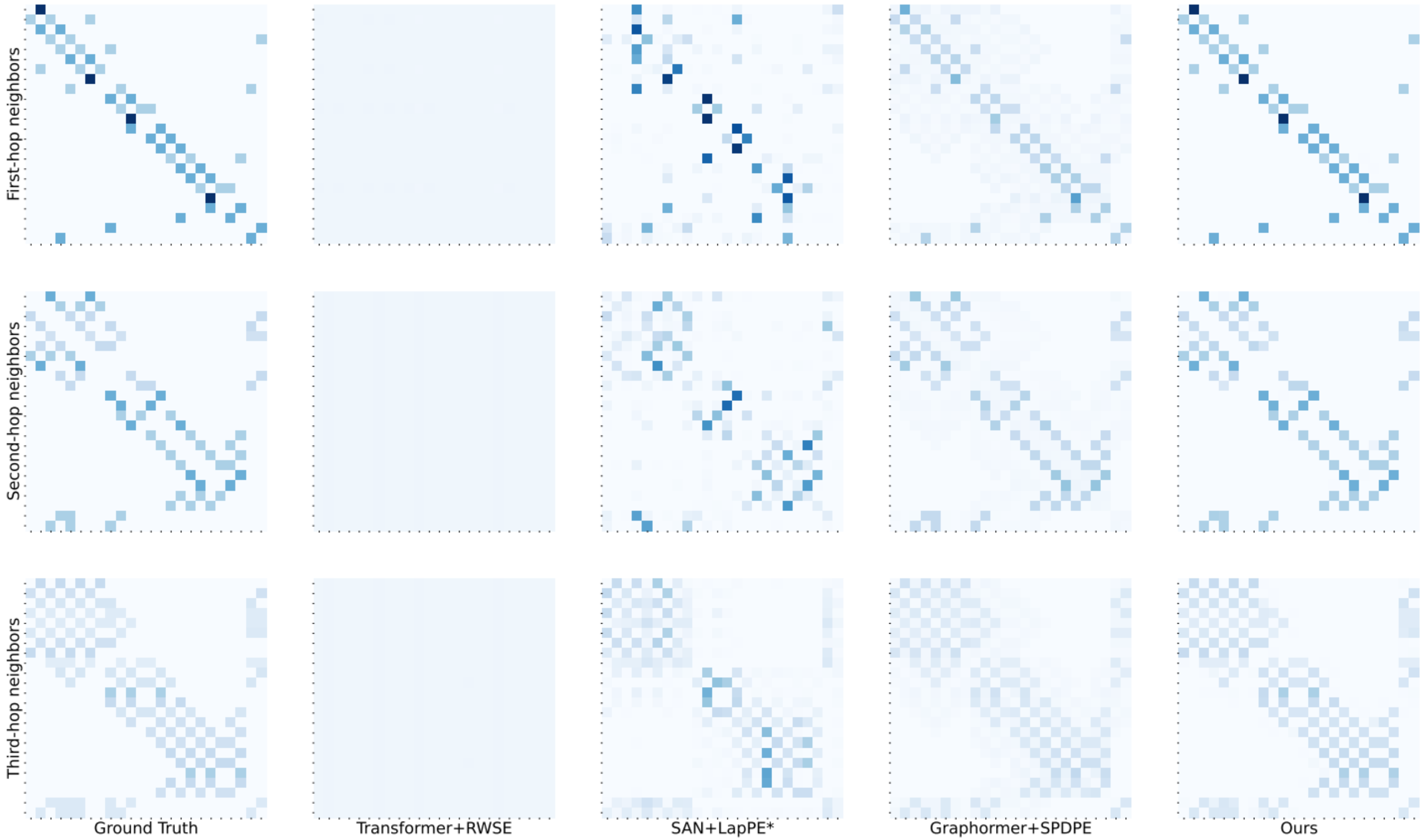


Others: Hyper-parameter settings

Hyperparameter	Peptides-func	Peptides-struct	PCQM4Mv2
# Transformer Layers	4	4	16
Hidden dim	96	96	256
# Heads	4	8	8
Dropout	0	0	0.1
Attention dropout	0.5	0.5	0.1
Graph pooling	mean	mean	mean
PE dim (walk-step)	17	24	16
PE encoder	linear	linear	linear
Batch size	32	32	256
Learning Rate	0.0003	0.0003	0.0002
# Epochs	200	200	150
# Warmup epochs	5	5	10
Weight decay	0	0	0
# Parameters	443,338	438,827	15.3M

Others: Synthetic Experiments

GRIT attention is successful at matching both the sparsity pattern and attention magnitudes of the target (far left)



Visualization of learned attention scores for the synthetic experiment on learning to attend to (k = 1, 2, 3)-hop neighbors

Others: Synthetic Experiments