Yale

Graph Transformers

CPSC483: Deep Learning on Graph-Structured Data

Rex Ying

Readings

- Readings are updated on the website (syllabus page)
- **Lecture 13 readings:**
  - Do Transformers Really Perform Bad for Graph Representation?
  - Recipe for a General, Powerful, Scalable Graph Transformer

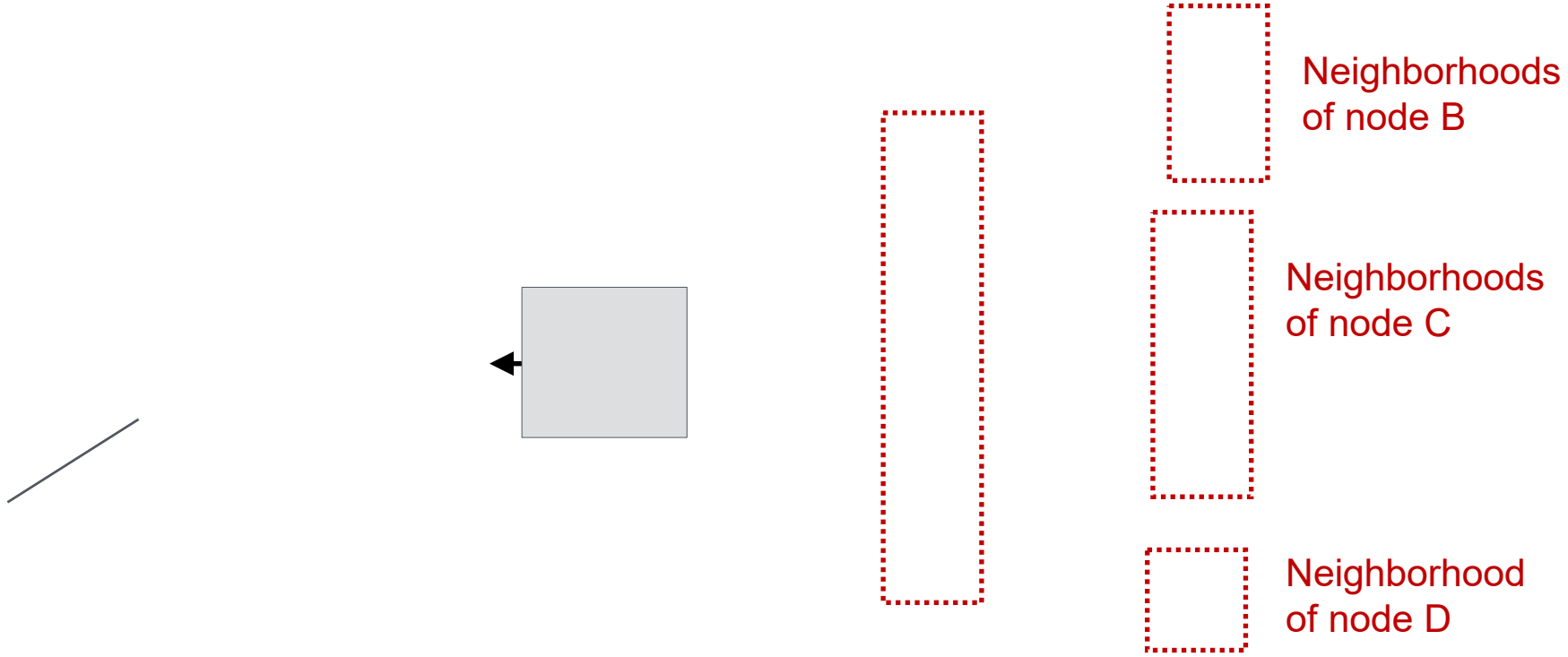
Content

- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- Scalability
- Empirical Verification

Content

- **Motivation of Graph Transformers**
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- Scalability
- Empirical Verification

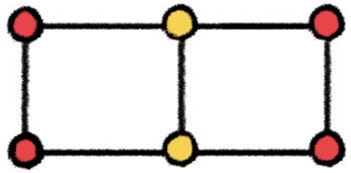
Recap: Graph Neural Networks

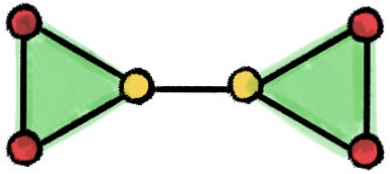


A single message-passing layer can only explore the relationship between node and its **one-hop** neighbors

Message Passing Drawbacks (1)

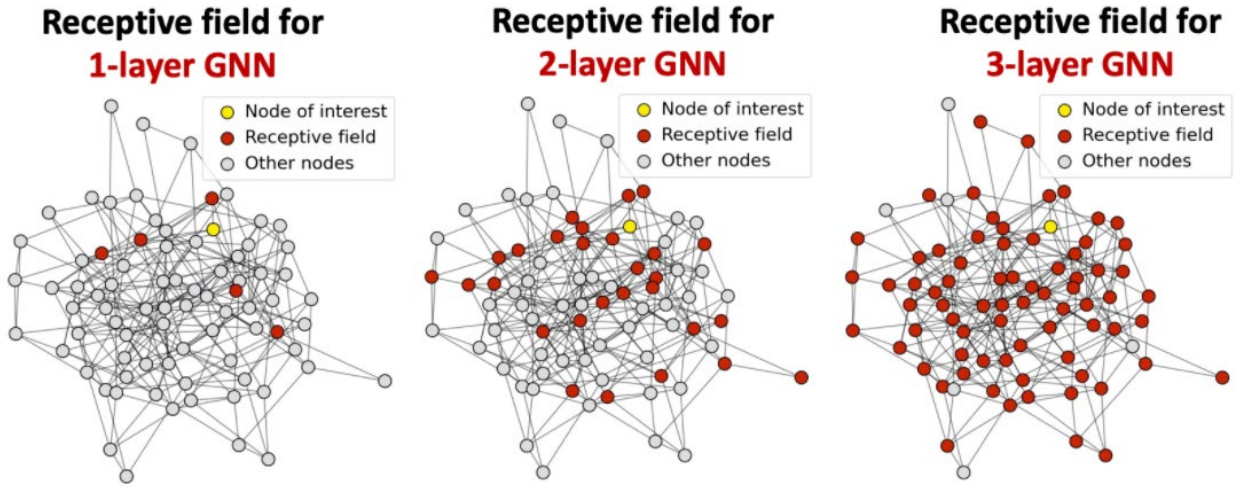
- **Expressiveness**: 1-order Message passing GNNs (**MPNN**) have limited expressiveness (at most as powerful as 1-WL test)





Non-isomorphic graphs that cannot be distinguished by MPNN

- **Over-smoothing:** node features tend to converge to the same value with the increasing number of message passing layers

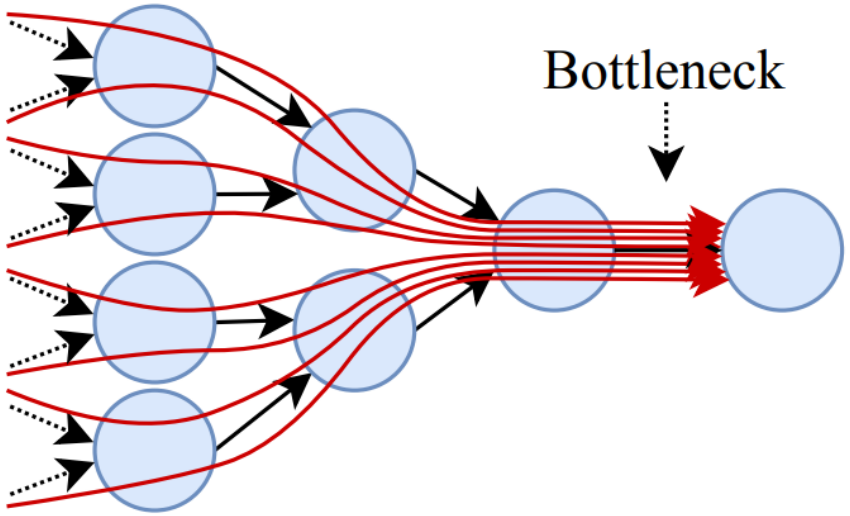


Receptive field quickly **covers the entire graph as the number of layers increases**

Message Passing Drawbacks (2)

- **Over-squashing:** In tasks relying on long-distance interactions, an **exponentially-growing** amount of information from distant nodes is squashed into a fixed-size vector.

- Message Passing Layers cannot **capture long-range dependencies effectively.**

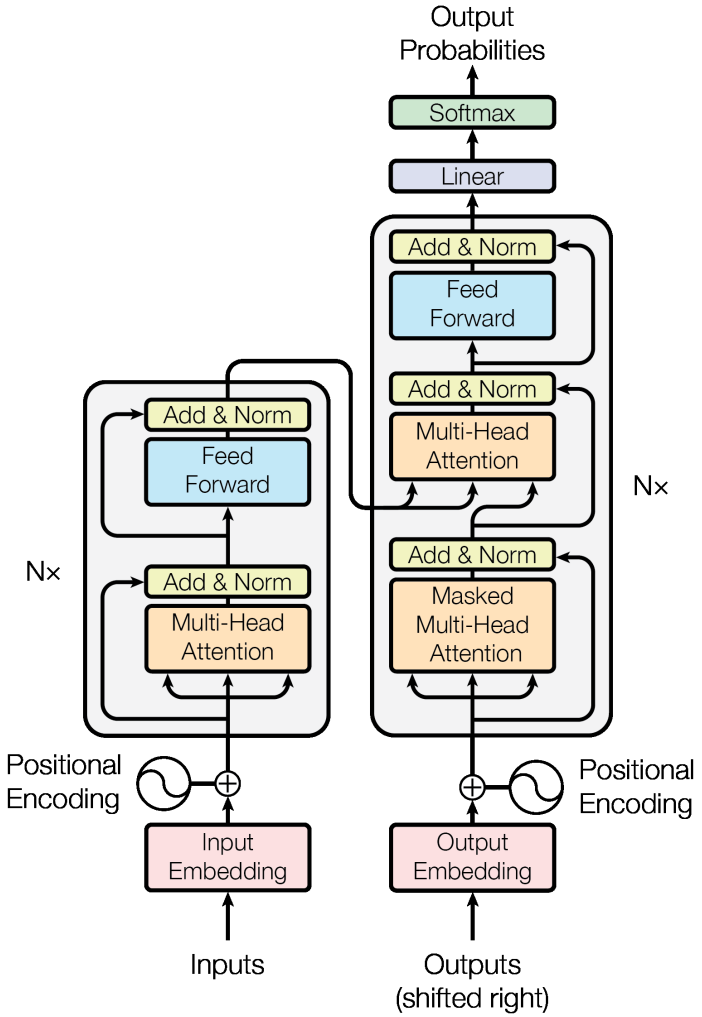


The bottleneck of graph neural networks

Alon, Uri, and Eran Yahav. "On the bottleneck of graph neural networks and its practical implications."

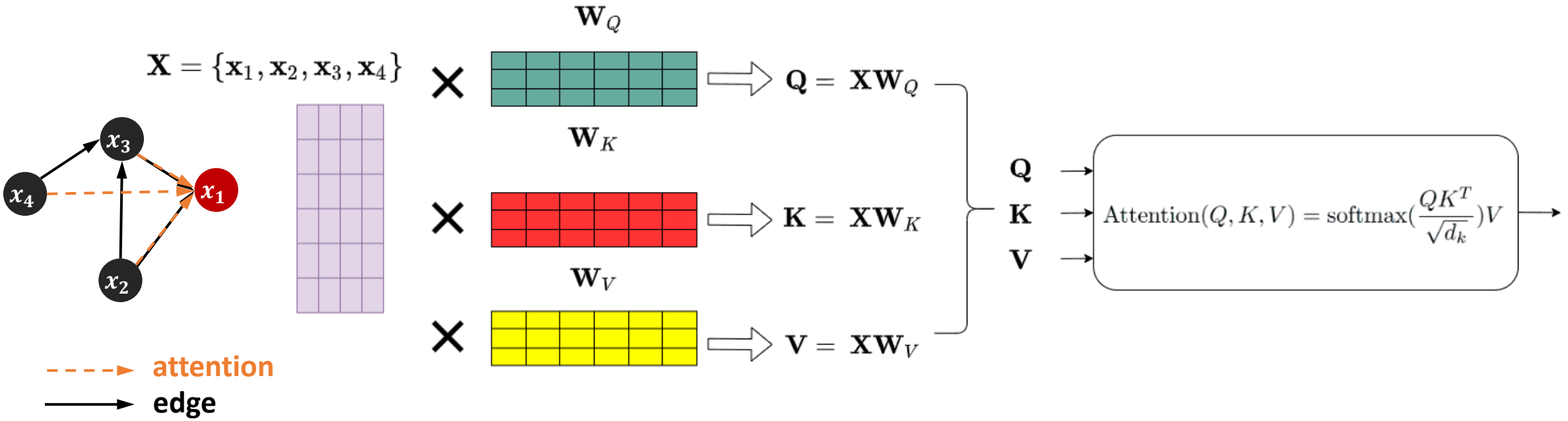
Motivation of Graph Transformer

- **Transformer is powerful in modelling sequential data**, such as natural language, speech, computer vision.
- Transformer **enables long-range connections** as all tokens attend to each other
- **Motivation**: Can Transformer architectures be used to model graphs and improve graph representation learning?
  - Expressiveness
  - Over-smoothing
  - Over-squashing (long-range)



Attention in Graph Transformers

Naïve full attention on node set:



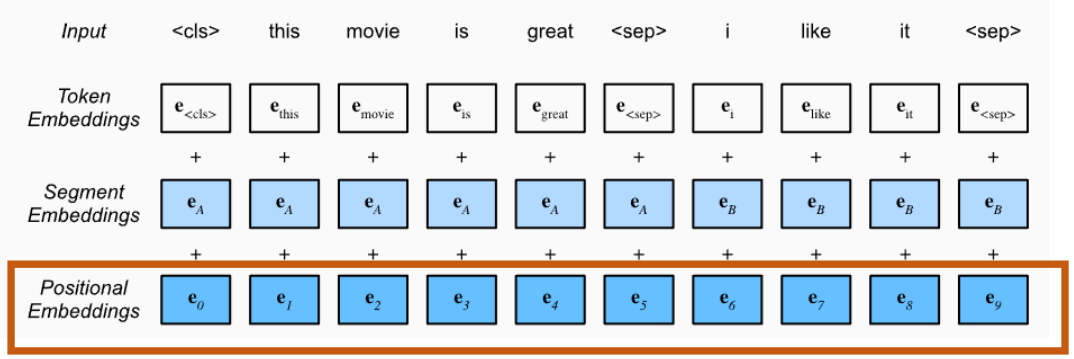
- Naturally capture **long-range dependencies**
- **Edge features** are not captured effectively

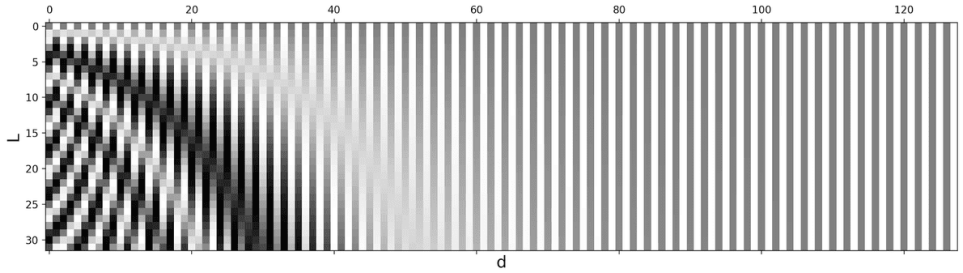
Content

- Motivation of Graph Transformers
- **Challenges for Building Graph Transformers**
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- Scalability
- Empirical Verification

Challenges for Graph Transformers (1)

- Language has natural sequential order, while graphs are permutation invariant to node ordering.
- Recall Positional Encoding in Lecture 8:





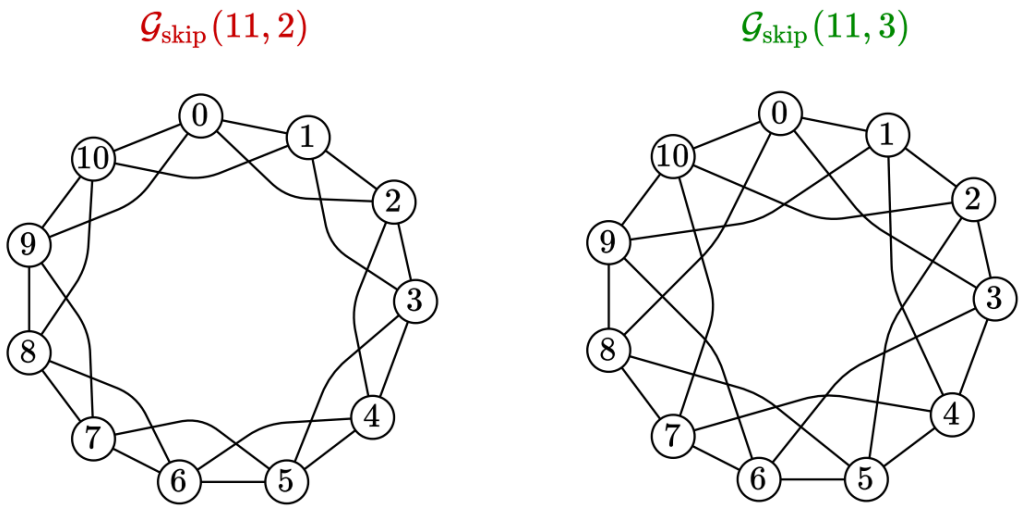
$$PE_{(\boldsymbol{pos},2\boldsymbol{i})} = \sin\left(\boldsymbol{pos}/10000^{2\boldsymbol{i}/d_{model}}\right)$$

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

- Positional Encoding for sequential input fails to identify nodes in a graph
- How to better **indicate the position of the node in a graph** ⇒ **Positional Encoding (PE)**

Challenges for Graph Transformers (2)

- Message passing GNNs suffer from **limited expressiveness** (1-WL test)
- Message passing GNNs cannot capture **local structure information** sufficiently.
- How to **better incorporate neighborhood information** in graph transformers ⇒ **Structural Encoding (SE)**



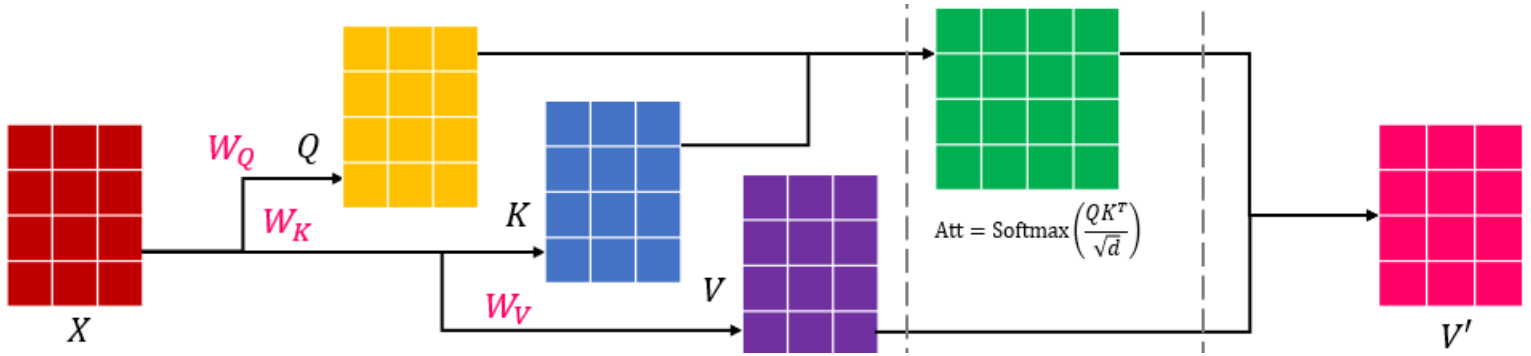
Circular Skip Links (CSL) Graphs

- Message passing GNNs **fail to distinguish Circular Skip Links (CSL)** graph pairs [1]
- **Structural Encoding** (which captures local substructure) can distinguish graph pairs[2]

[1] Murphy, Ryan, et al. "Relational pooling for graph representations." *International Conference on Machine Learning*. PMLR, 2019.
[2] Rampášek, Ladislav, et al. "Recipe for a general, powerful, scalable graph transformer." *Advances in Neural Information Processing Systems* 35 (2022): 14501-14515.

Challenges for Graph Transformers (3)

- Computational complexity of MPNN: $O(E)$, where E is the number of edges
- Conventional Transformers induce square computational complexity $O(N^2)$ in the number of nodes N $\Rightarrow$ **improve the scalability of graph transformers**
  - The $\boldsymbol{QK}^T$ matrix multiplication, $\mathrm{Softmax}()$, $\mathrm{Att}V$ value updates **in full attention** all consume $\boldsymbol{n^2}$ time and memory.



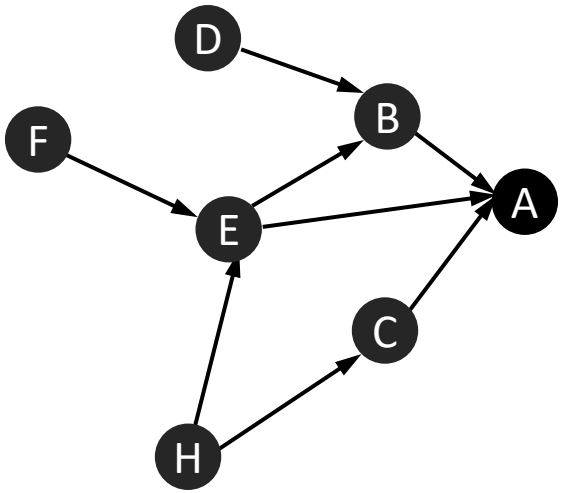
Complexity: $O(Nd d_X)$ $O(N^2 d)$ $O(N^2 d)$

Content

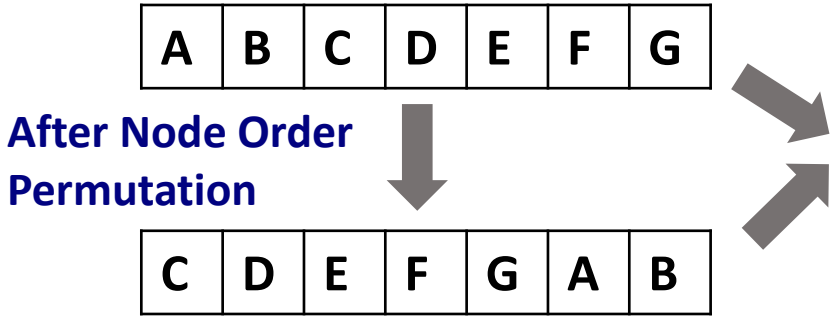
- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- **Encodings: Positional & Structural**
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- Scalability
- Empirical Verification

Why traditional PE fails?

- Graphs **do not have a natural node ordering** like sequences.
- **Permutation equivariance** should be preserved by positional encoding.



No natural node ordering within a graph



Should have the same output

Differences between PE and SE

Positional Encodings (PE)

- Provide **an embedding of the position of a given node within the graph**
- **Assumption:** when two nodes are close to each other within a graph or subgraph, their PE should also be close.

Structural Encodings (SE)

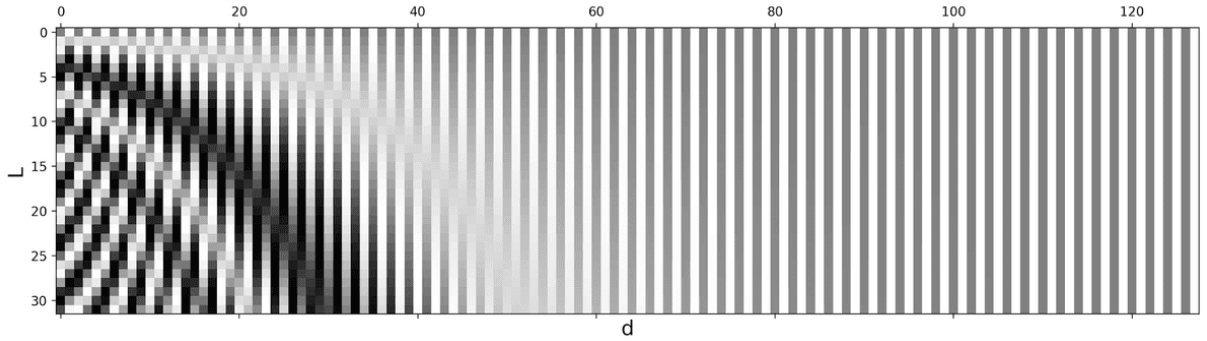
- Provide **an embedding of the structure of neighborhoods or subgraphs** to help increase the expressiveness and the generalizability of GNNs
- **Assumption:** when two nodes share similar subgraphs, or when two graphs are similar, their SE should also be close.

Content

- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - **Laplacian Positional Encoding**
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- Scalability
- Empirical Verification

Why Sin/Cos as Positional Encoding (1)

- Recall the positional encoding for sequential input:



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

- **Question**: Why we use Sine/Cosine functions as positional encoding?
- In Euclidean space, **sine/cosine functions are the eigenfunctions of the Laplacian operator** f, i.e., $Lf = \lambda f$ with some λ.

Definition of Laplacian Operator:
$$\boldsymbol{Lf} = \mathbf{div}(\boldsymbol{\nabla f})$$

Why Sin/Cos as Positional Encoding (2)

- **Example**: in $\mathbb{R}^2$, $\sin(mx + ny)$, $\cos(mx + ny)$ are eigenfunctions of L, with $\lambda = -(m^2 + n^2)$.
- **Theorem [Fourier Series]**: any square-integrable function $F(x, y)$ on $\mathbb{R}^2$, i.e.,

$$\iint_{-\infty}^{\infty} |F(x, y)|\, dx\, dy < \infty$$

can be expanded into

$$F(x, y) = \sum_{m,n=-\infty}^{+\infty} a_{m,n} \sin(mx + ny) + b_{m,n} \cos(mx + ny)$$

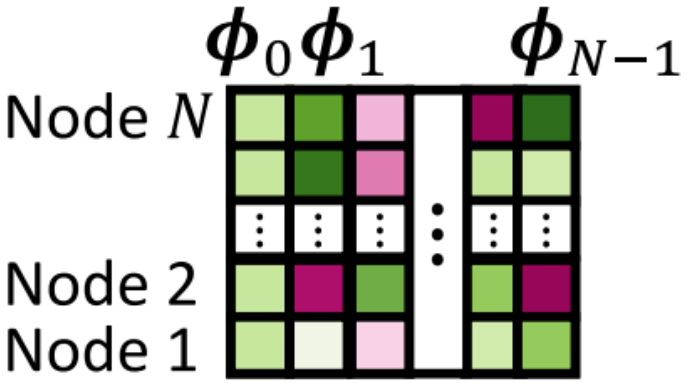
where $a_{m,n}$ and $b_{m,n}$ are Fourier coefficients

Laplacian in Graph Domain

- In graph domain, the eigenvectors of the graph Laplacian naturally encode the positional information of the given graph

$$\mathrm{L} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U^T \Lambda U$$

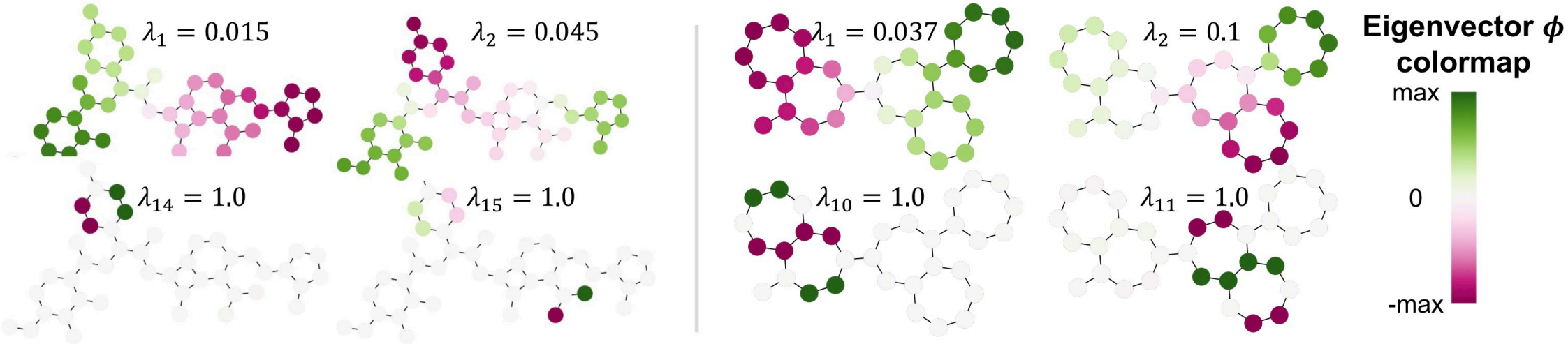
- $U = [\phi_0, \ldots, \phi_{N-1}]^T$, where ϕ_i indicates the i-th eigenvector



- **Each column represents an eigenvector**
- **One row per node.**

Eigenvectors Reflect Graph Substructures

- Eigenvectors can be used to discriminate between different graph structures and sub-structures



Kreuzer, Devin, et al. "Rethinking graph transformers with spectral attention." *Advances in Neural Information Processing Systems* 34 (2021): 21618-21629.

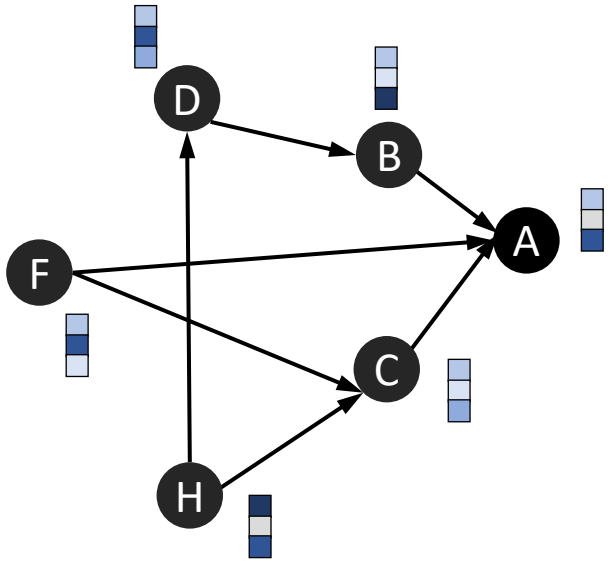
The **low-frequency** eigenvectors ϕ_1, ϕ_2 are **spread across the graph**

The **high-frequency** eigenvectors ϕ_i $(i > 10)$ **resonate in local structures**

Laplacian Positional Encoding (1)

- Use Laplacian eigenvectors as node positional encoding (usually select m eigenvectors with **m-lowest eigenvalues**)
- The Laplacian positional Encoding **for the *j*-th node**:

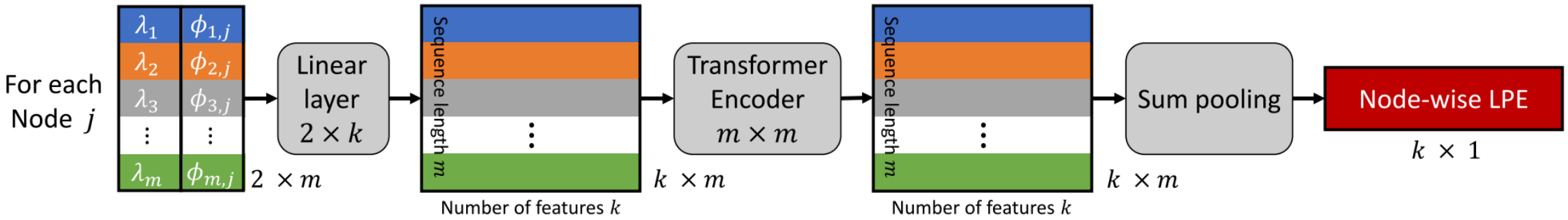
$$p_j^{\text{LapPE}} = [\phi_{1j}, \phi_{2j}, \ldots, \phi_{mj}]$$



Node-level Positional Encoding
(associated with each node)
$(m = 3)$

Laplacian Positional Encoding (2)

- Advance version [1]:
  - concatenate with the corresponding eigenvalues
  - learn the potential positional encoding by neural networks



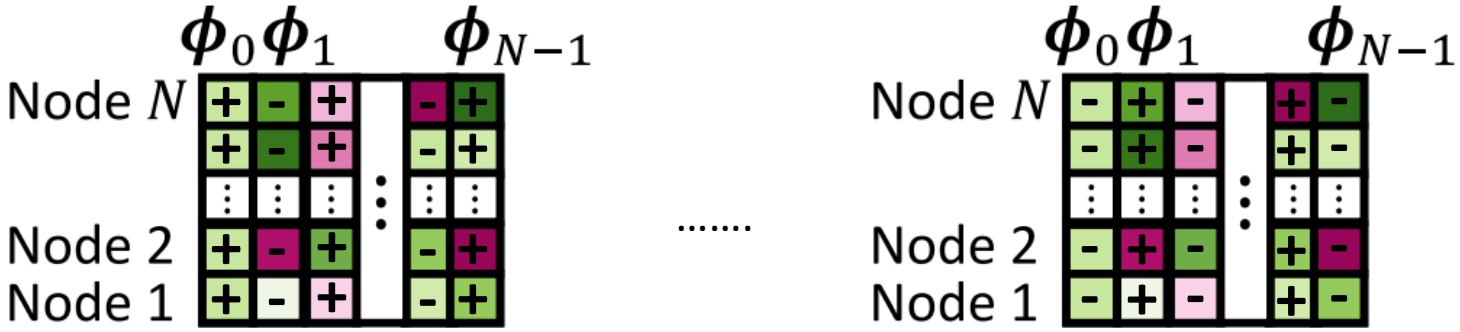
LPE: **L**earned **P**ositional **E**ncoding

[1] Kreuzer, Devin, et al. "Rethinking graph transformers with spectral attention." *Advances in Neural Information Processing Systems* 34 (2021): 21618-21629.

Sign Ambiguity in LapPE

- If ϕ_i is an eigenvector, then $-\phi_i$ is also an eigenvector.
- There is a total of $\mathbf{2^m}$ possible sign combinations when choosing $\boldsymbol{m}$ **eigenvectors of a graph**

  - Example:



8 possible Laplacian Positional Encodings in total

Solutions to Sign Ambiguity

- **Randomly flipping the sign** of eigenvectors during training to guarantee the sign invariance [1]
- Applying **permutation equivariant mapping** over ϕ_i and $-\phi_i$ [2] :

$$\mathrm{PE} = [f(\phi_1) + f(-\phi_1), \ldots, f(\phi_m) + f(-\phi_m)]$$

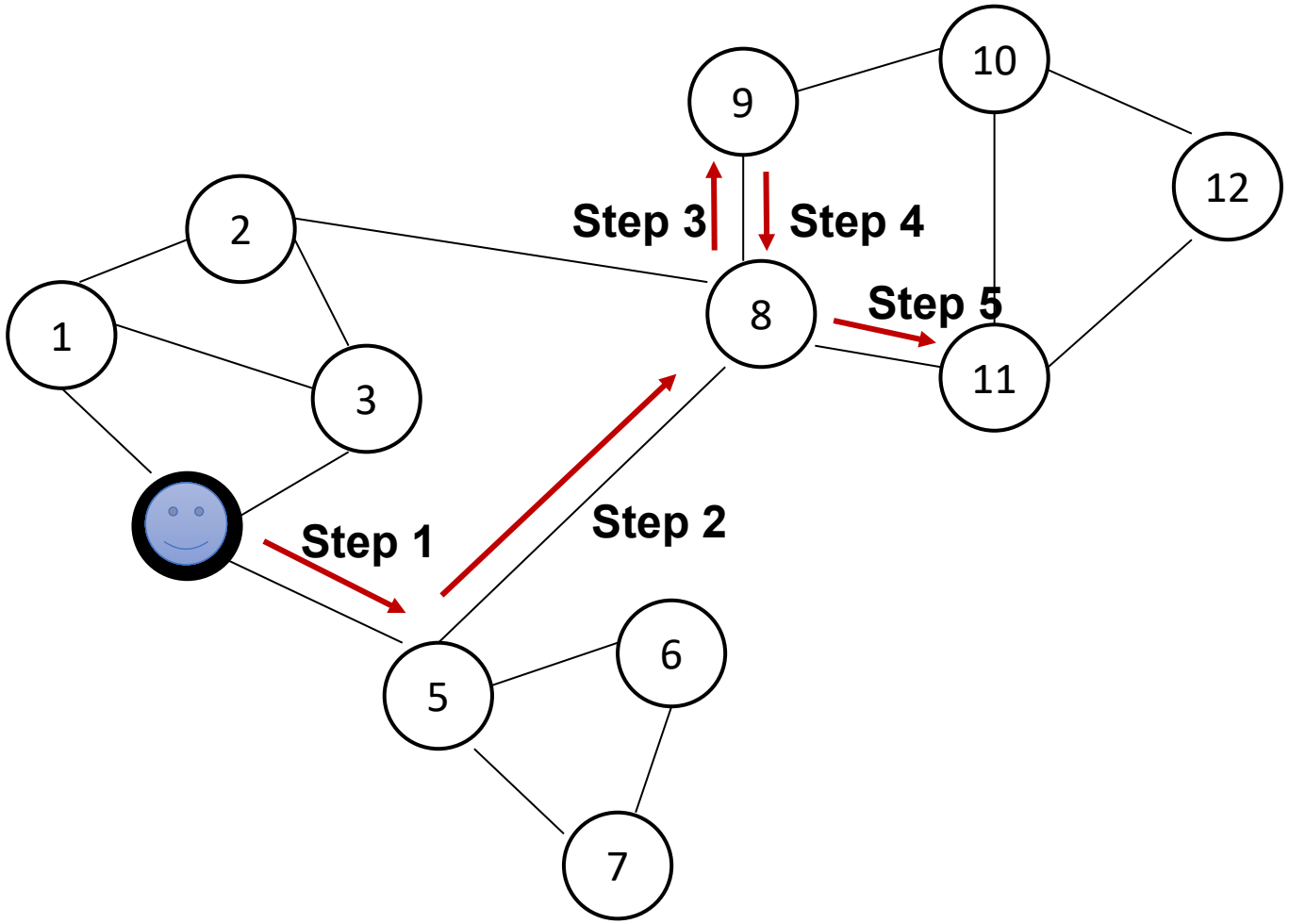
  - $f: \mathbb{R}^N \to \mathbb{R}^N$ can be any permutation equivariant networks, such as DeepSets or GNNs
  - Node Embedding Matrix $\mathrm{PE} \in \mathbb{R}^{N \times m}$
  - N is the number of nodes

[1] Dwivedi, Vijay Prakash, et al. "Benchmarking graph neural networks." *arXiv preprint arXiv:2003.00982* (2020).
[2] Lim, Derek, et al. "Sign and basis invariant networks for spectral graph representation learning." *arXiv preprint arXiv:2202.13013* (2022).

Content

- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - **Random Walk Structural Encoding**
- Token Construction
- Forward Propagation
- Scalability
- Empirical Verification

Random Walk



- Given a *graph* and a *starting point*, we **select a neighbor** of it at **random**
- Move to this neighbor
- Select a neighbor of this point at random, and move to it.
- The (random) sequence of points visited this way is a **random walk on the graph**.

Random Walk Encoding

- $\mathrm{RW} = \mathrm{AD}^{-1}$ is the **random walk operator**. A is the adjacency matrix, D is the degree matrix.
- **Random Walk Encoding (RWE)** for the i-th node is defined with k-steps of random walk as

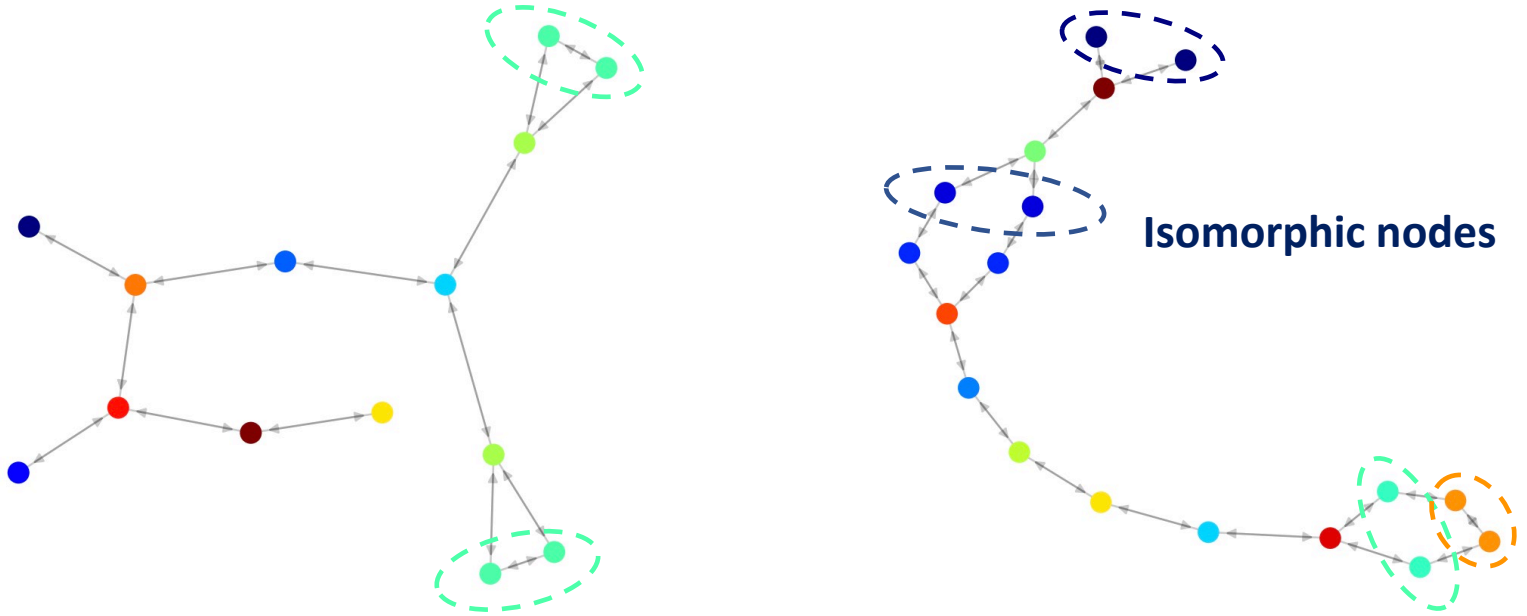
$$p_i^{RWE} = [\mathrm{RW}_{ii}, \mathrm{RW}_{ii}^2, \dots, \mathrm{RW}_{ii}^k] \in \mathbb{R}^k$$

- RWE encodes the landing probability of a node to itself in 1 to k steps of random walk ⇒ meaningful higher-order structure information!
- Note: RWE is a Structural Encoding.

Question:
1. What happens when k increases? (Higher-order neighborhood is considered)
2. What happens to the RWE if a node is densely connected to its neighborhood?

Dwivedi, Vijay Prakash, et al. "Graph neural networks with learnable structural and positional representations."

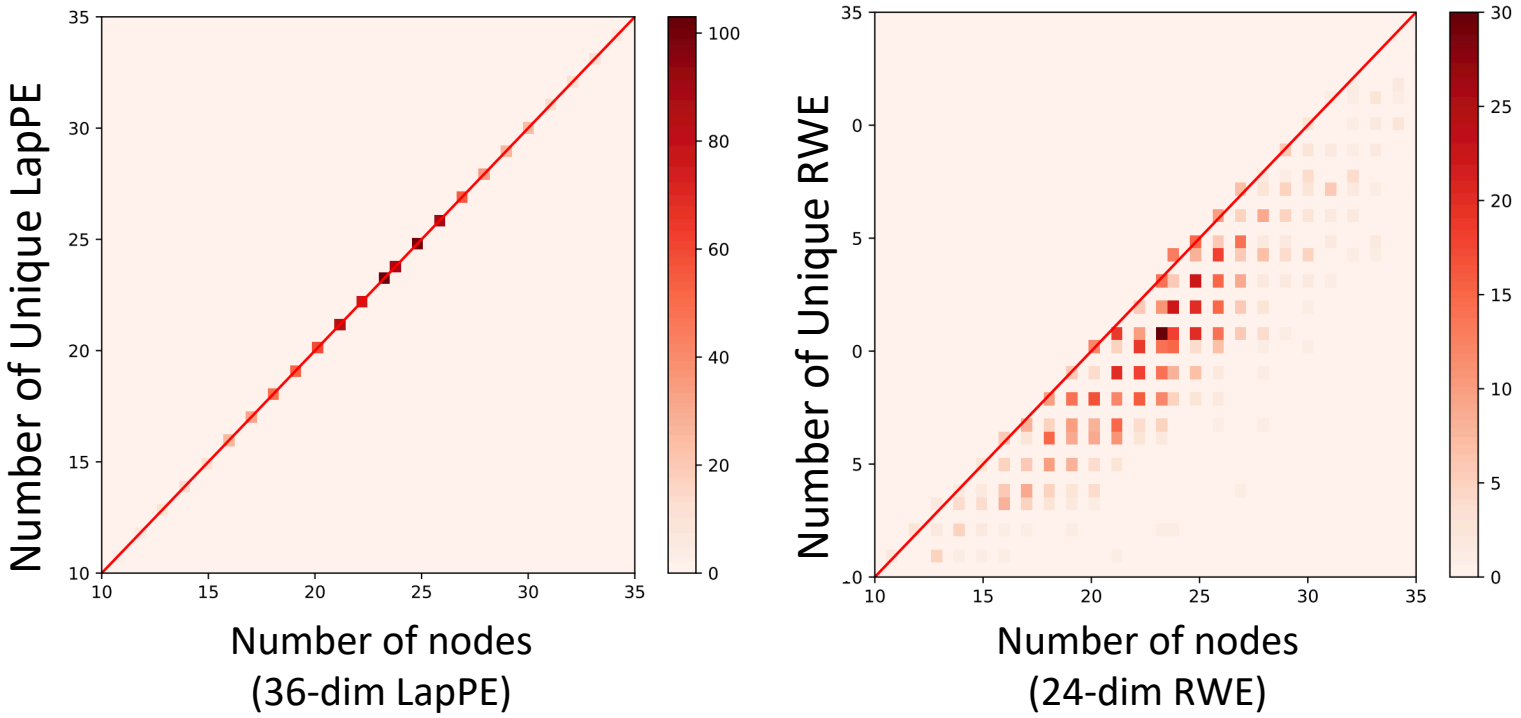
Random Walk Encoding Visualization



- **Different node color** represents a **unique RWE** vector with $\mathrm{k} = 24$
- The nodes with the same color / RWE are isomorphic in the graph, i.e. their k-hop structural neighborhoods are the same!

Comparison between RW and Laplacian

LapPE and RWE on ZINC validation set:



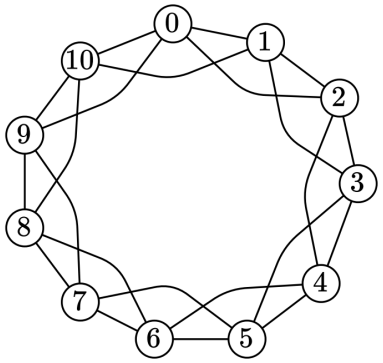
Color darkness indicates the number of graphs

- Laplacian PE guarantees unique node representations better (an injective mapping)
- With RWE, most graphs have a large amount of nodes with unique positional encoding
- Note: RWE do not suffer from the sign ambiguity of LapPE ⇒ more efficient training

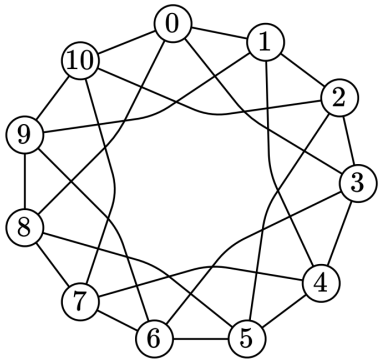
PE Improves Expressiveness

- Positional Encoding can distinguish graph pairs that cannot be correctly distinguished by Message-passing GNNs (1-WL algorithm)

$\mathcal{G}_{\text{skip}}(11, 2)$



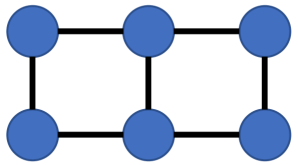
$\mathcal{G}_{\text{skip}}(11, 3)$



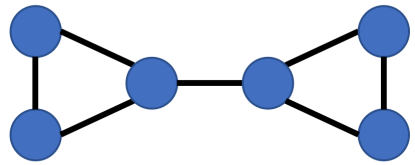
$$\begin{bmatrix} RW_{ii}^1 \\ RW_{ii}^2 \\ RW_{ii}^3 \\ RW_{ii}^4 \end{bmatrix} \quad \begin{bmatrix} 0.0000 \\ 0.2500 \\ 0.0938 \\ 0.1406 \end{bmatrix} \quad \begin{bmatrix} 0.0000 \\ 0.2500 \\ 0.0000 \\ 0.1719 \end{bmatrix}$$

4-dimensional RWE distinguish CSL graph pairs

Graph 1



Graph 2

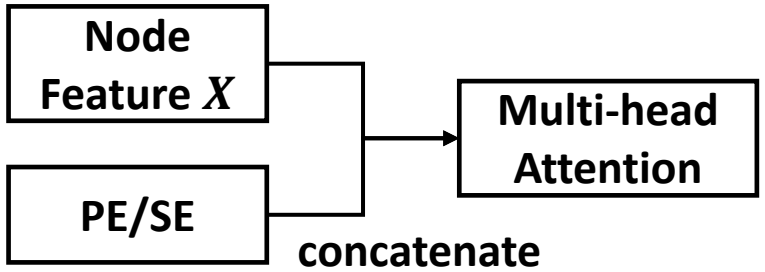


Eigenvalues of Graph 1	Eigenvalues of Graph 2
0	0
1	0.438
2	3
3	3
3	3
5	4.562

non-isomorphic graphs that can be distinguished by their eigenvalues but not by Message Passing GNNs

How to inject PE/SE into (1)

- **Concatenate with node/edge/graph features** before Attention layers
  - Example: SAN, GPS



- Inject PE/SE with **attention score**
  - Example: GraphiT

$$\text{PosAttention}(Q, V, K_r) = \text{normalize}\left(\exp\left(\frac{QQ^\top}{\sqrt{d_{\text{out}}}}\right) \odot K_r\right) V \in \mathbb{R}^{n \times d_{\text{out}}}$$

Graph kernel, where $K_r(i, j)$ encodes relative position between nodes i and j

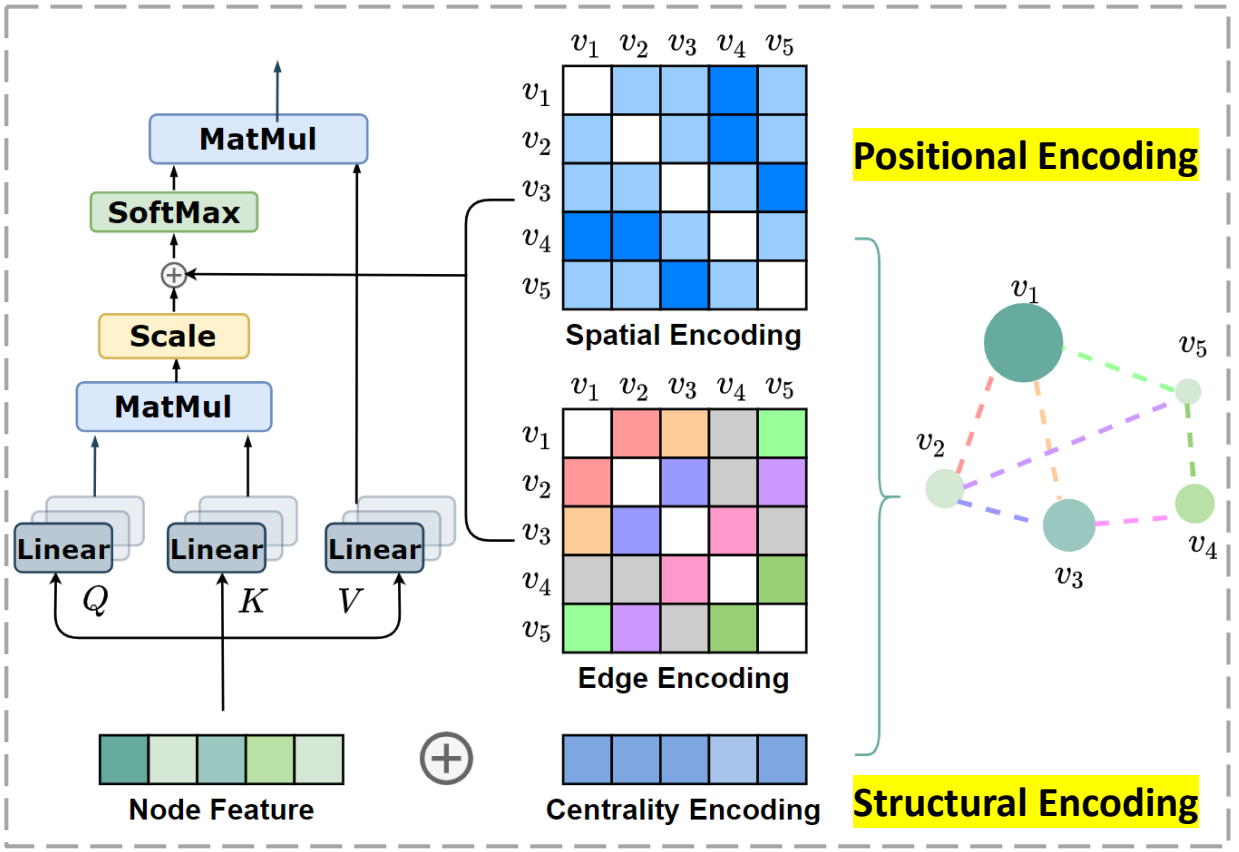
How to inject PE/SE (2)

- Inject local PE/SE with **node inputs** and treat relative PE/SE as additional **attention bias**
  - Example: Graphormer

Attention: $e_{ij} = \frac{(\mathbf{h}_i \mathbf{W}_q)(\mathbf{h}_j \mathbf{W}_k)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)} + c_{ij}$

Spatial Encoding:
Shortest path between v_i, v_j

Edge Encoding:
Average all edge features along the shortest path between v_i, v_j $c_{ij} = \frac{1}{N} \sum_{e \in SP(i,j)} x_e w_e$

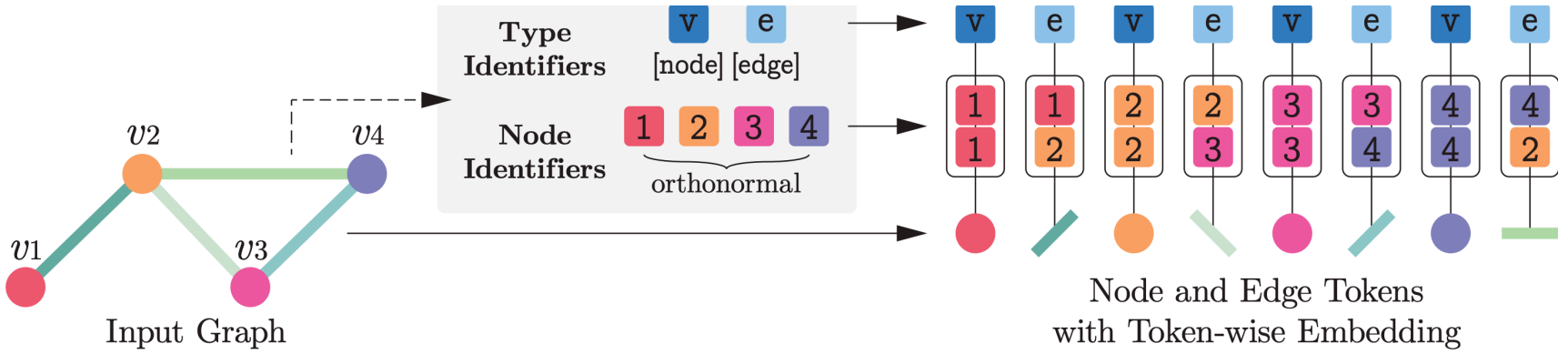


Content

- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- **Token Construction**
- Forward Propagation
- Scalability
- Empirical Verification

Token Construction (1)

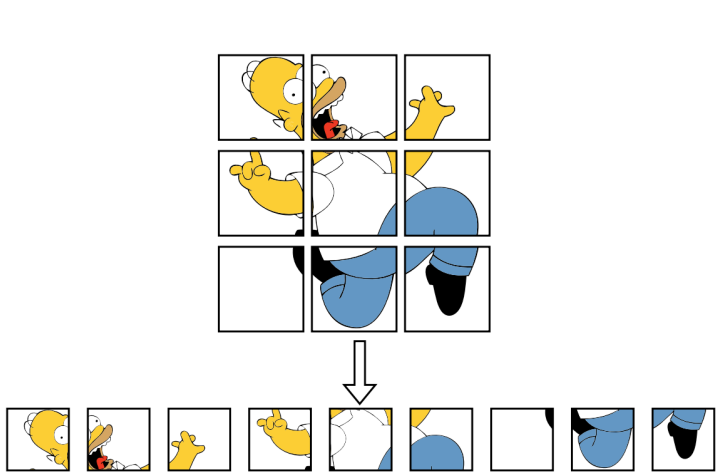
- Using nodes-only as input tokens is the most common approach
  - The complexity is $O(N^2)$ (conventional graph transformers with full attention)
- Use nodes and edges as input tokens (Examples: EGT, TokenGT)
  - Model higher-oder node-edge and edge-edge interactions ⇒ **stronger expressiveness**
  - The complexity is $O(N+E)^2$



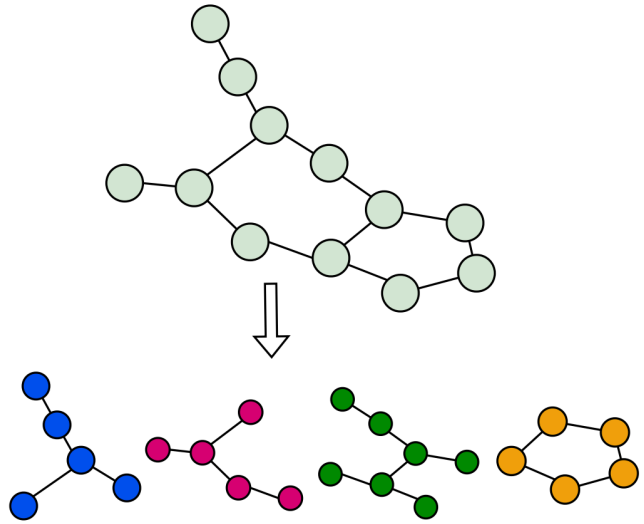
Kim, Jinwoo, et al. "Pure transformers are powerful graph learners." *Advances in Neural Information Processing Systems* 35 (2022): 14582-14595.

Token Construction (2)

- Use **subgraphs** as tokens (Example: CoarFormer, MLP-Mixer)
  - A natural generalization of ViT to graph domain
  - Significantly reduces the computational complexity
  - Enable graph transformers to scale to large graphs



Images



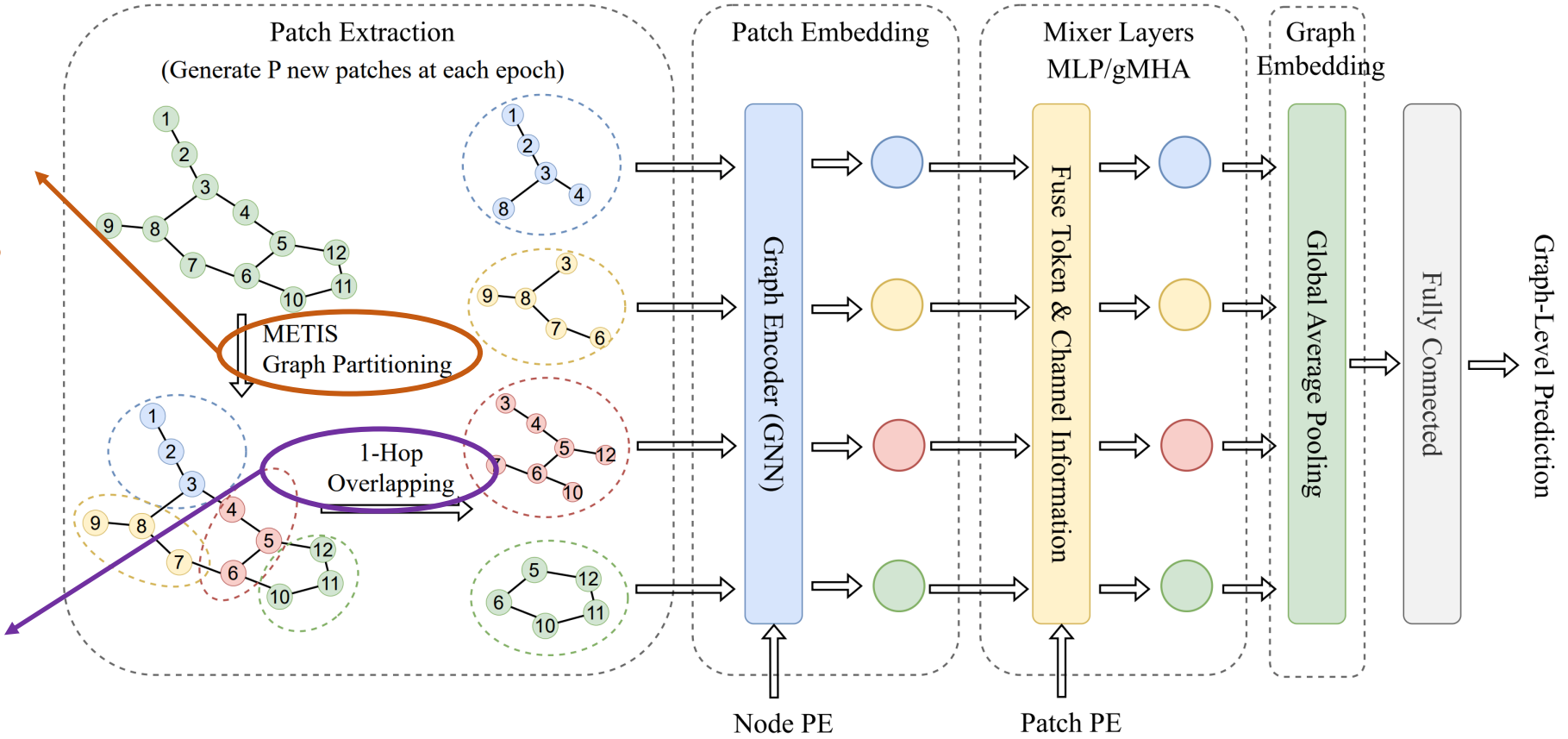
Graphs

MLP-Mixer Architecture

METIS:

- graph partitioning algorithm
- # intra-cluster links is much higher than inter-cluster links

- Involve **all 1-hop neighbors** to capture important edge information
- e.g., the cutting edges



Patch PE counts the number of connecting edges between cluster $\mathcal{V}_i$ and cluster $\mathcal{V}_j$

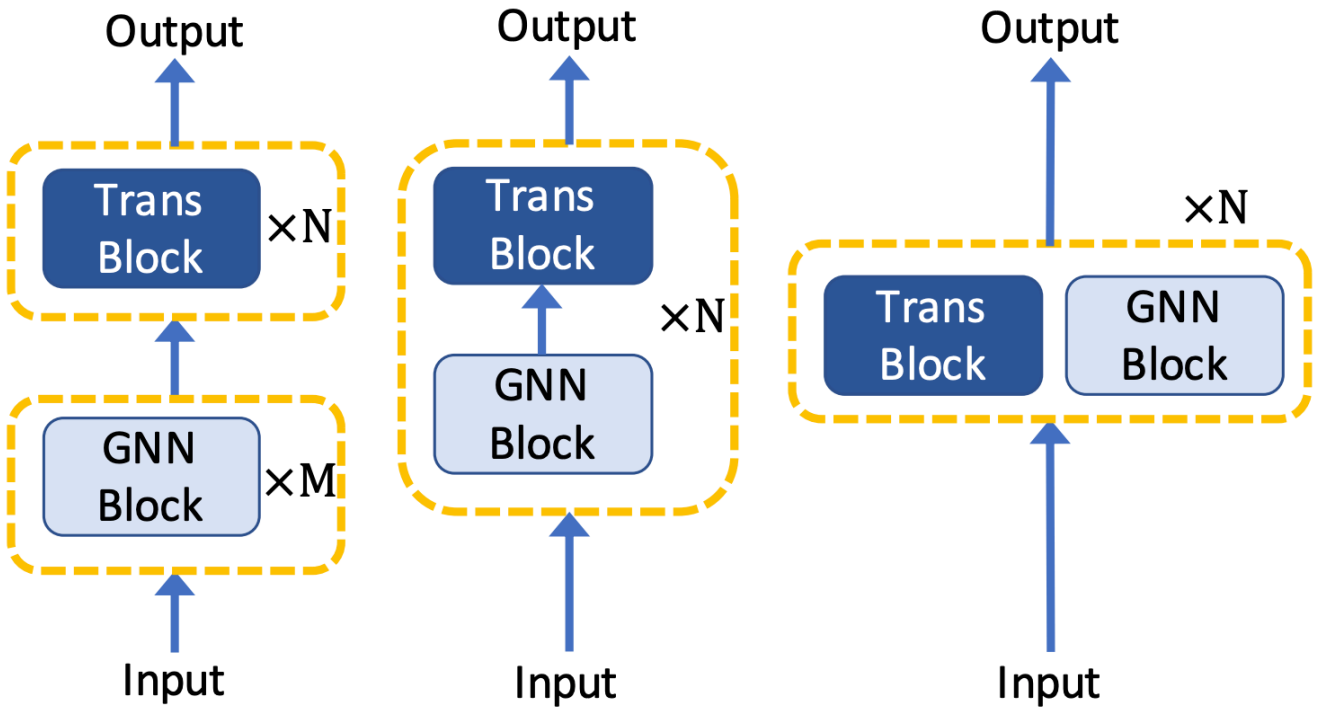
$$A^P_{ij} = |\mathcal{V}_i \cap \mathcal{V}_j|$$

Content

- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- **Forward Propagation**
- Scalability
- Empirical Verification

Forward Propagation

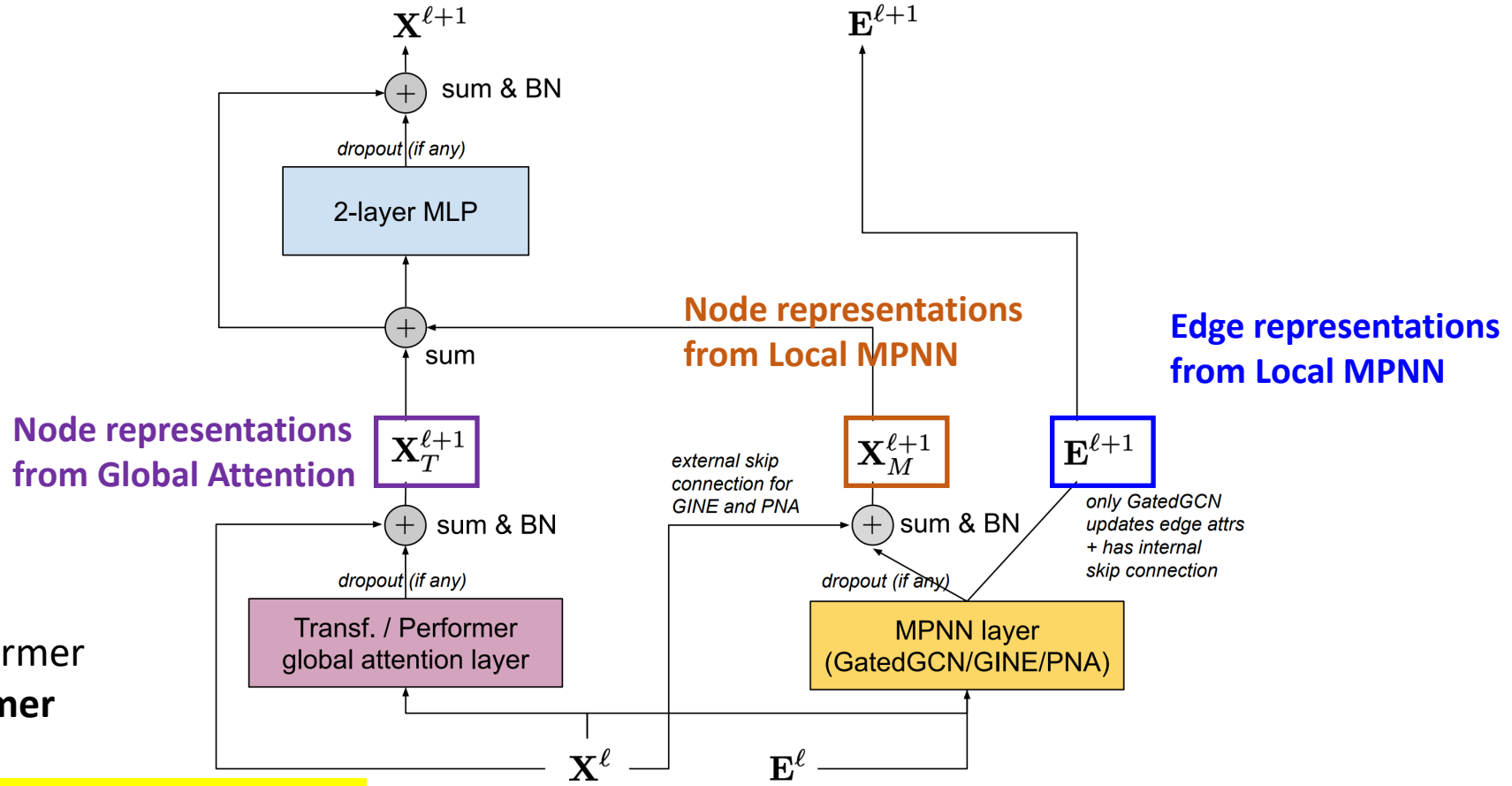
- GNNs can be used as **auxiliary modules with transformer** architectures



(a) Before Trans (b) Alternatively (c) Parallel

General, Powerful, Scalable (GPS) layers

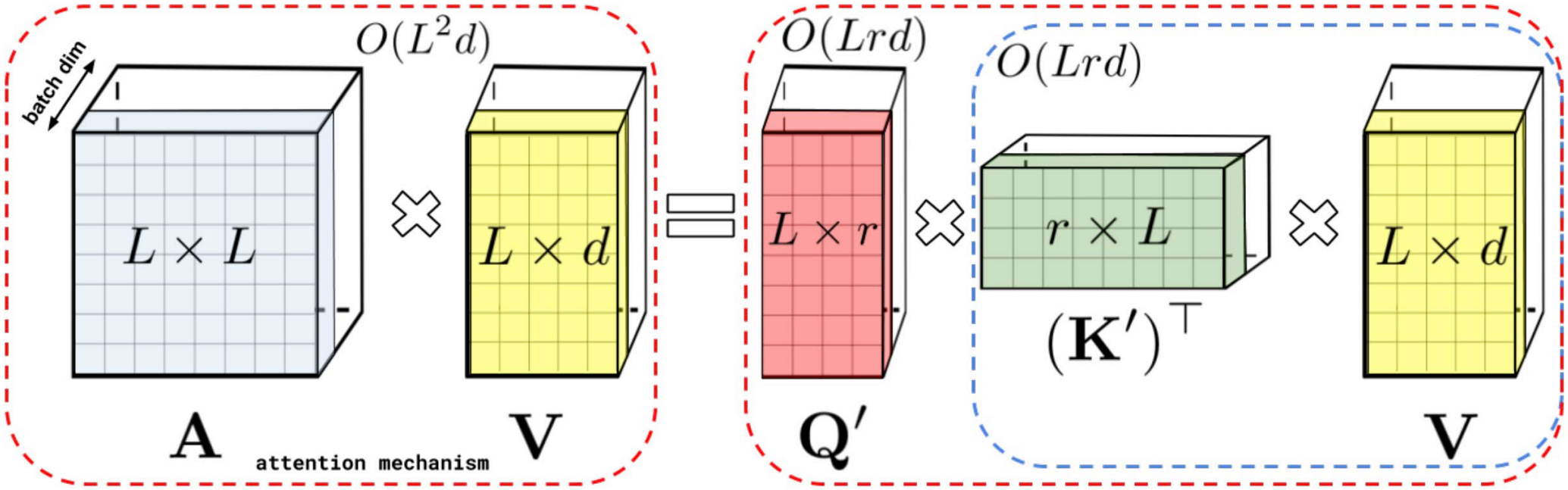
- GPS Layer combines local MPNN and global attention blocks parallelly



👉 Transformer
Performer
BigBird
(Recall sparse transformers in Lec.8)

Performer in GPS

- Main idea: **kernelize** the softmax in Transformer's self-attention and **change order of matrix multiplication** to reduce complexity



Step1: $(\mathrm{K}')^{\mathrm{T}}\mathrm{V}$
Step2: $\mathrm{Q}'(\mathrm{K}')^{\mathrm{T}}\mathrm{V}$
⇒ **Linear attention w.r.t. # nodes**

- Note: other linear attention mechanisms (e.g., BigBird) can also be used to accelerate full attention.

Choromanski, Krzysztof, et al. "Rethinking attention with performers." *arXiv preprint arXiv:2009.14794* (2020).

Experimental Results of Graph Transformers

method	#param.	test MAE
GIN [54]	509,549	0.526±0.051
GraphSage [18]	505,341	0.398±0.002
GAT [50]	531,345	0.384±0.007
GCN [26]	505,079	0.367±0.011
GatedGCN-PE [4]	505,011	0.214±0.006
MPNN (sum) [15]	480,805	0.145±0.007
PNA [10]	387,155	0.142±0.010
GT [13]	588,929	0.226±0.014
SAN [28]	508, 577	0.139±0.006
$\text{Graphormer}_{\text{SLIM}}$	489,321	**0.122**±0.006

Dataset: ZINC

method	#param.	AUC (%)
GCN-GraphNorm [5, 8]	526K	78.83±1.00
PNA [10]	326K	79.05±1.32
PHC-GNN [29]	111K	79.34±1.16
DeeperGCN-FLAG [30]	532K	79.42±1.20
DGN [2]	114K	79.70±0.97
GIN-VN[54] (fine-tune)	3.3M	77.80±1.82
Graphormer-FLAG	47.0M	**80.51**±0.53

Dataset: OGB-MolHIV

Graph Transformers (e.g., Graphormer) surpass message passing GNNs generally

Content

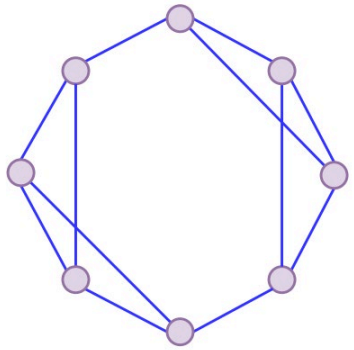
- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- **Scalability**
- Empirical Verification

Scalability

- Standard (dense) transformers have **quadratic complexity** in the number of tokens, which limits their scalability to extremely long sequences.
- Possible reason: linear attention mechanisms are **designed for natural sequences** (e.g., language, images)
- Question: how to design **specialized attention** mechanisms that are more tailored to learning interactions on **general graph**?

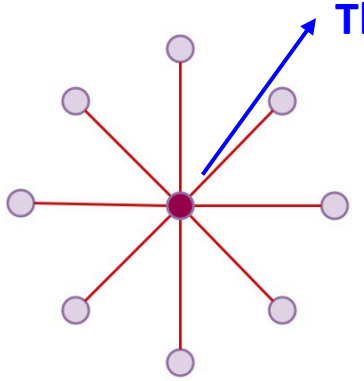
Exphormer: Sparse Transformers for Graphs

- Key point: design a ***graph-centric*** **sparse attention** mechanism that makes use of the underlying structure of the input graph



Expander graph
with degree 3

 # edges: $O(N)$
(will come back)



Global attention
with a virtual node

 # edges: $O(N)$

Random Expander Graph

d-regular Expander Graph Generation

- Given a node set V, pick $d/2$ permutations $\pi_1, \pi_2, \dots, \pi_{d/2}$ on V, each π_i are chosen independently and uniformly from all possible permutations
- Then, generate edge sets:

$$E = \left\{ \left(i, \pi_j(i)\right), \left(i, \pi_j^{-1}(i)\right) \,\middle|\, j \in \left\{1,2,\dots,\frac{d}{2}\right\}; i \in \{1,2,\dots,|V|\} \right\}$$

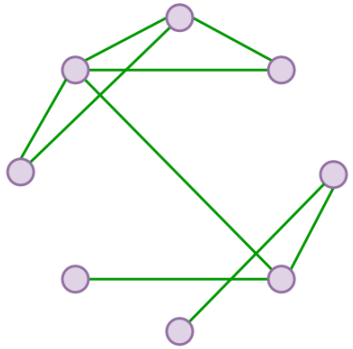
- **The number of edges is $d|V|$. The degree of each node is d.**

***Theorem** [1]: For a sparse attention mechanism generated by a d-regular expander on n nodes, **stacking $O(\log n)$ transformer layers** models full attention.*

[1] Shirzad, Hamed, et al. "Exphormer: Sparse transformers for graphs." *arXiv preprint arXiv:2303.06147* (2023).

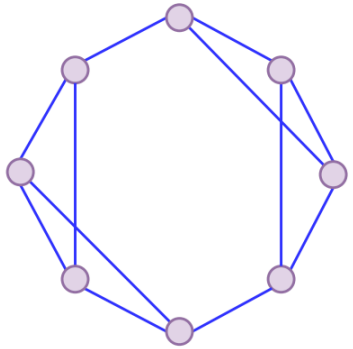
Expander: Long-range Dependencies

- **The expander graph** and **virtual node** allow propagating information between pairs of nodes that are distant in the input graph G without connecting all pairs of nodes.



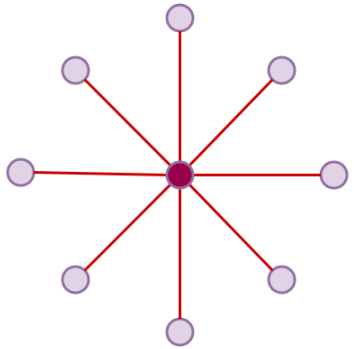
(a)

Local attention
(i.e., real edges of the graph)
 # edges: $O(E)$



(b)

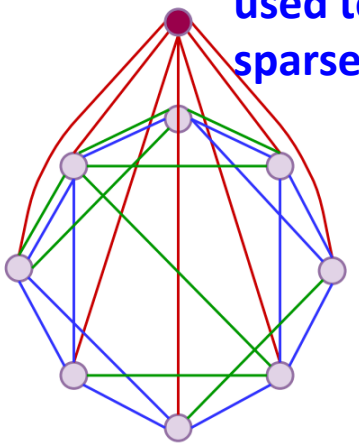
Expander graph
with degree 3
 # edges: $O(N)$



(c)

Global attention
with a virtual node
 # edges: $O(N)$

used to create sparse attention



(d)

Combination of all attention components
 # edges: $O(N+E)$

Experimental Results

- The complexity of the combination attention is $O(N + E)$, N, E are numbers of nodes and edges.

Edges in the original graph are significant!

The effect of virtual nodes and expander graphs depends on the dataset.

Dataset	No Local Edges	No Expander Edges	No Global Nodes	All Components
Cifar10	74.62 ± 0.12	74.53 ± 0.19	74.68 ± 0.19	$\mathbf{74.69 \pm 0.13}$
Malnet-Tiny	92.64 ± 0.55	$\mathbf{94.02 \pm 0.21}$	92.48 ± 0.33	92.06 ± 0.18
Pattern	86.59 ± 0.03	86.70 ± 0.02	86.14 ± 0.08	$\mathbf{86.74 \pm 0.02}$
PascalVOC-SP	0.3708 ± 0.0039	0.3588 ± 0.0013	$\mathbf{0.3975 \pm 0.0037}$	0.3682 ± 0.0042
Peptides-Struct	0.2631 ± 0.0007	$\mathbf{0.2481 \pm 0.0007}$	0.2655 ± 0.0003	0.2643 ± 0.0008
Computer	90.34 ± 0.45	91.48 ± 0.41	$\mathbf{91.59 \pm 0.31}$	91.43 ± 0.53

Content

- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- Scalability
- **Empirical Verification**

Empirical Verification (Expressiveness)

Model	Easy	Medium		Hard
	EDGES	TRIANGLES-SMALL	TRIANGLES-LARGE	CSL
	2-way Accuracy ↑	**10-way Accuracy ↑**	**10-way Accuracy ↑**	**10-way Accuracy ↑**
GIN	**98.11 ±1.78**	71.53 ±0.94	33.54 ±0.30	10.00 ±0.00
Transformer	55.84 ±0.32	12.08 ±0.31	10.01 ±0.04	10.00 ±0.00
Transformer (LapPE)	**98.00 ±1.03**	78.29 ±0.25	10.64 ±2.94	**100.00 ±0.00**
Transformer (RWSE)	97.11 ±1.73	**99.40 ±0.10**	**54.76 ±7.24**	**100.00 ±0.00**
Graphormer	97.67 ±0.97	**99.09 ±0.31**	42.34 ±6.48	90.00 ±0.00

EDGES: Predict if an edge connects two nodes in the graph

TRIANGLES: count the number of triangles in the graph
LARGE: train/val graphs are much smaller than test graphs

Circular Skip Links Graphs

- Graph Transformers with structural bias generally perform well on all three tasks with a few exceptions.
- The shortest-path encoding in Graphormer distinguishes 9 out of the 10 classes correctly in CSL dataset
- All graph transformers **generalize poorly to larger triangle dataset** ⇒ **still suffer from limited expressiveness**

Empirical Verification (Oversmoothing)

Benchmarking on six graph datasets that especially suffer from the over-smoothing issue of GNNs:

Model (PE/SE type)	ACTOR	CORNELL	TEXAS	WISCONSIN	CHAMELEON	SQUIRREL
Geom-GCN [Pei *et al.*, 2020]	31.59 ±1.15	60.54 ±3.67	64.51 ±3.66	66.76 ±2.72	**60.00 ±2.81**	38.15 ±0.92
GCN (no PE/SE)	33.92 ±0.63	53.78 ±3.07	65.95 ±3.67	66.67 ±2.63	43.14 ±1.33	30.70 ±1.17
GCN (LapPE)	34.30 ±1.12	56.22 ±2.65	65.95 ±3.67	66.47 ±1.37	43.53 ±1.45	30.80 ±1.38
GCN (RWSE)	33.69 ±1.07	53.78 ±4.09	62.97 ±3.21	69.41 ±2.66	43.84 ±1.68	31.77 ±0.65
GCN (DEG)	33.99 ±0.91	53.51 ±2.65	66.76 ±2.72	67.26 ±1.53	46.36 ±2.07	34.50 ±0.87
$\text{GPS}^{\text{GCN+Transformer}}$ (LapPE)	37.68 ±0.52	66.22 ±3.87	75.41 ±1.46	74.71 ±2.97	48.57 ±1.02	35.58 ±0.58
$\text{GPS}^{\text{GCN+Transformer}}$ (RWSE)	36.95 ±0.65	65.14 ±5.73	73.51 ±2.65	**78.04 ±2.88**	47.57 ±0.90	34.78 ±1.21
$\text{GPS}^{\text{GCN+Transformer}}$ (DEG)	36.91 ±0.56	64.05 ±2.43	73.51 ±3.59	75.49 ±4.23	52.59 ±1.81	42.24 ±1.09
Transformer (LapPE)	**38.43 ±0.87**	**69.46 ±1.73**	**77.84 ±1.08**	76.08 ±1.92	49.69 ±1.11	35.77 ±0.50
Transformer (RWSE)	**38.13 ±0.63**	**70.81 ±2.02**	**77.57 ±1.24**	**80.20 ±2.23**	49.45 ±1.34	35.35 ±0.75
Transformer (DEG)	37.39 ±0.50	**71.89 ±2.48**	**77.30 ±1.32**	**79.80 ±0.90**	56.18 ±0.83	**43.64 ±0.65**
Graphormer (DEG only)	36.91 ±0.85	68.38 ±1.73	76.76 ±1.79	77.06 ±1.97	54.08 ±2.35	**43.20 ±0.82**
Graphormer (DEG, attn. bias)	36.69 ±0.70	68.38 ±1.73	76.22 ±2.36	77.65 ±2.00	53.84 ±2.32	**43.75 ±0.59**

Transformer: disabling the local GCN in GPS layers
DEG: using node degree as positional encoding

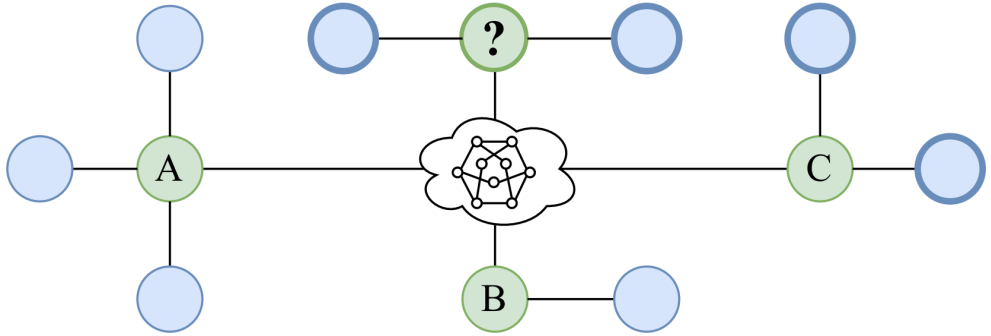
1. Geom-GCN is specialized for over-smoothing issue

2. PE/SE have minimal effect on GCN's performance

3. Global attention of a transformer empirically facilitates more successful information propagation

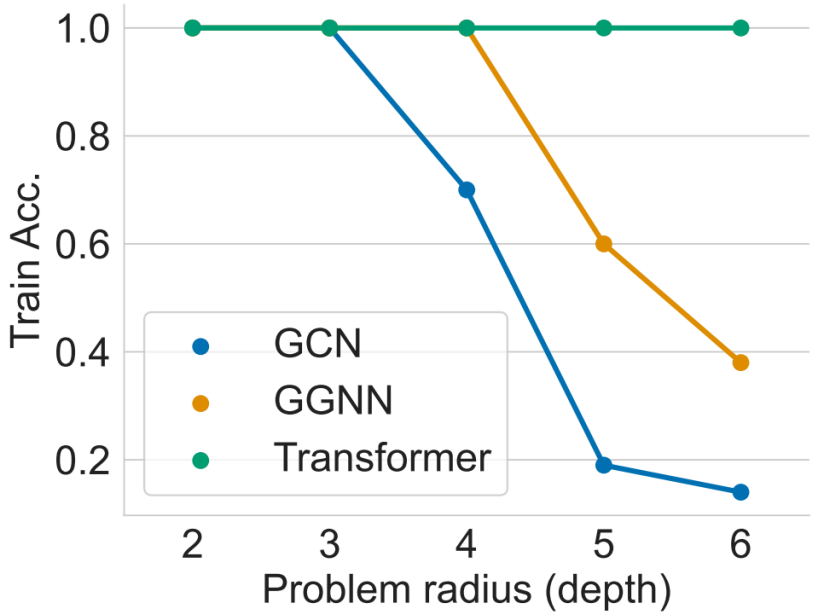
Empirical Verification (Long-range Dependencies)

- GNNs poorly capture long-range dependencies
- Neighbors-Match synthetic dataset:



A: three blue neighbors;
B: one blue neighbors;
C: two blue neighbors;
Require long-range information to predict the target

A tree graph of depth d
$d \uparrow$, more challenging



Graph Transformers have better ability to model **long-range dependencies** and help to **circumvent the over-squashing issue.**

Summary

- Graph Transformer helps to improve the **expressiveness**, alleviate **over-smoothing** and **over-squashing** issues.
- Challenges for graph transformer: positional encoding, structure encoding, scalability.
- Two typical encodings for positions and structures: **LapPE** and **RWE**
- Better PE/SE improves expressiveness.
- Token construction: node-only, node and edge, subgraph (a solution to extremely large-scale graphs)
- GNNs can be used as **auxiliary modules with transformer** architectures.