Beyond Sparse Graphs: Graph Transformers

Jiaxuan You
Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall
https://ulab-uiuc.github.io/CS598/

Logistics: Updated Submission Task

- **1 DDL:** The deadline for the submission task is **Nov 21 (Thu), 11:59 PM CT**. Please plan the progress of your project reasonably.
  - We will kick off the peer-review session right after the submission. We will also give instructions on review and response on Nov 22 (Fri).
  - The fall break begins from Nov 23 (Sat). Enjoy your break after submitting your paper :)
- **2 Length:** We expect a minimum length of **6 pages** in ICLR 2025 format for the **draft submission**.
  - For the draft version, you are expected to include at least sections such as related work, methods, and experiment settings.
  - We will use the OpenReview to receive submissions, which will be announced on Canvas by this weekend.

Logistics: Updated Submission Task

- **3 Grading:** The submission task counts towards **15% (writing) + 15% (implementation) = 30%** of your final grade.
  - For the 15% of writing, **only 5%** is determined by the **draft version** (**due on Nov 21**) and **10%** is determined by the **final version** (**due on Dec 8**).
  - The 15% of implementation is determined by the code you provide for the **final version** (**due on Dec 8**).
  - You can revise your draft version during the review and response as well as the presentation stage based on others' feedback.
  - However, we encourage you to complete as much as possible for your submission to receive more comprehensive feedback.

Logistics: Updated Schedule

12	Nov 13 Wed	No class	Paper Writing		
	Nov 15 Fri	GNN applications: graph mining (remote)	Paper Writing		Assignment 4, due on Nov 17
13	Nov 20 Wed	GNN applications: science	Paper Writing		Submission task, due on Nov 21 (only draft required)
	Nov 22 Fri	Conclusion	Review & Response	Review & response task, out	
14	Nov 27 Wed	No class (fall break)	Review & Response		
	Nov 29 Fri	No class (fall break)	Review & Response		

- **No class and office hour on Nov 13 (Wed)**
- **Remote session on Nov 15 (Fri)**
  - Zoon link will be announced on Canvas and Slack.

Logistics: Coding Homework

- **Assignment 3 due**
  - Please submit your code and written answers to Canvas.
  - The submission deadline is **Nov 3 (Sun) 11:59 PM, CT**.
- **Assignment 4 out**
  - Assignment will be released on Canvas today.
  - Implement graph transformers
  - Submit your code and written answers downloaded from Colab to Canvas by **Nov 17 (Sun) 11:59 PM, CT**.

Today's Lecture

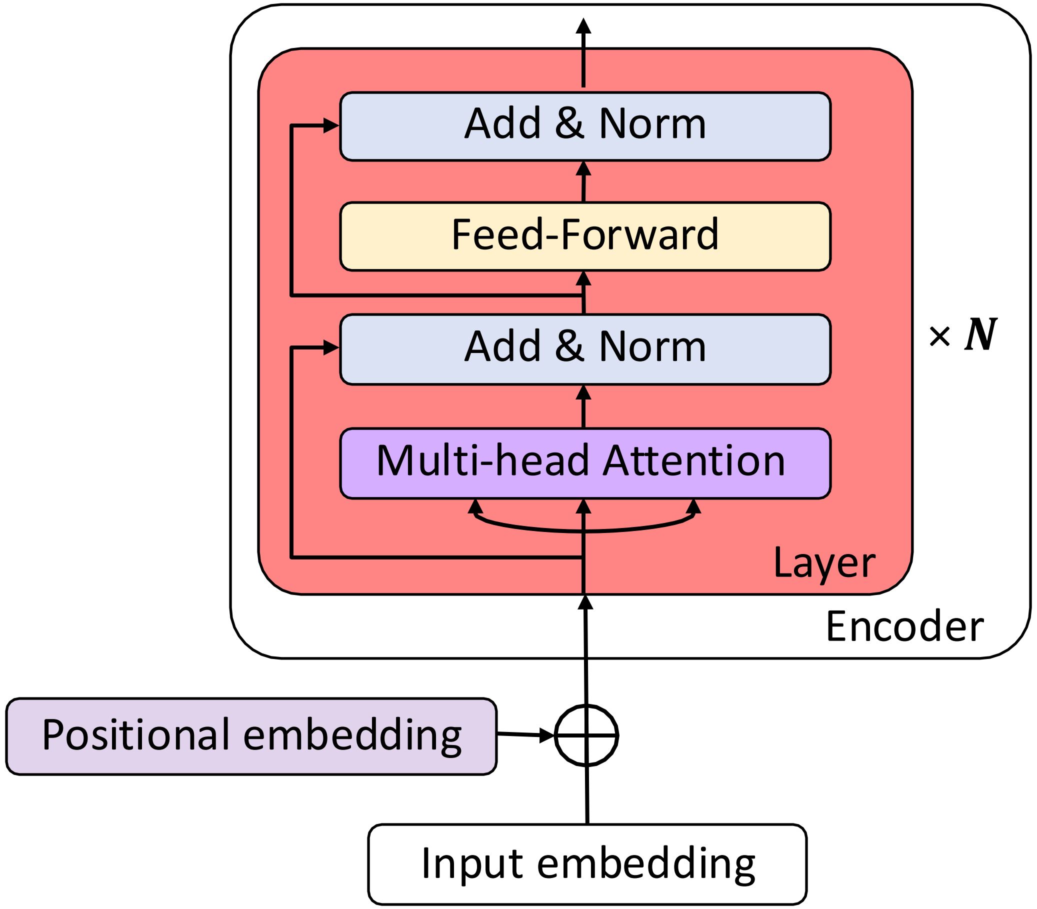
- Graph attention is **closely related** to transformers
- Graph learning techniques can **make transformers more efficient**
- Transformer can **inspire GNN architectures**

Beyond Sparse Graphs: Graph Transformers

Self-Attention and Transformers

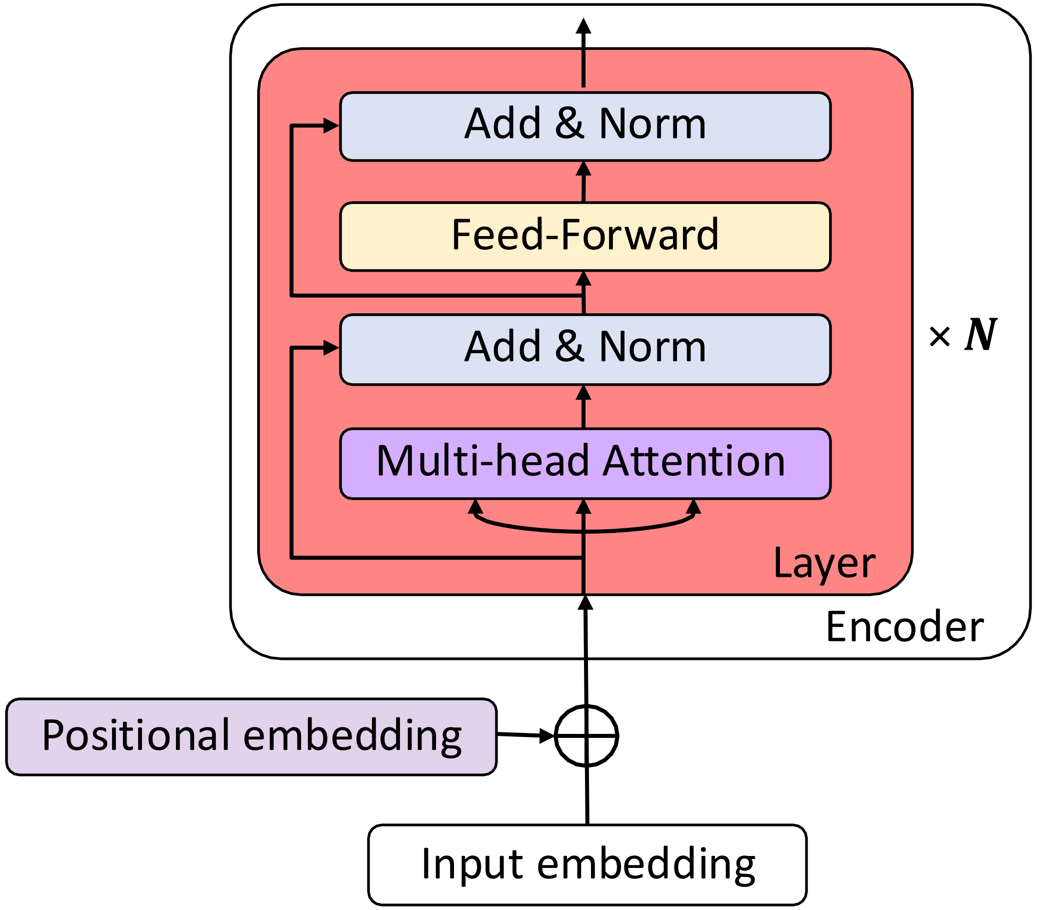
Transformers: Overview

- **Original paper**: Attention is all you need [Vaswani et al., 2017].
- **Key component**: Multi-head self-attention
- **Other components** of a transformer layer: layer normalization, skip connection, position- wise feed-forward layer (FFN, or MLP)



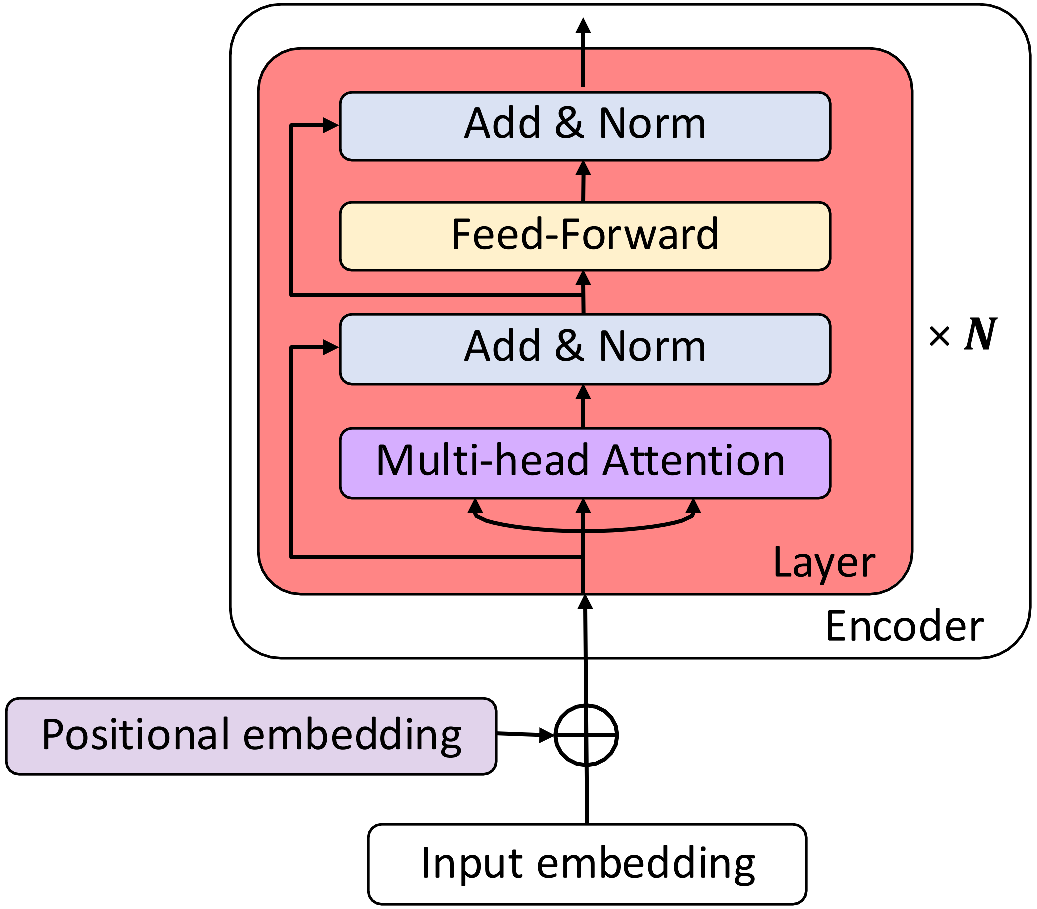
Transformers: Overview

- **Model usage**: Pre-training followed by fine- tuning. The transferred model can be:
  - Encoder-only (e.g BERT)
    - Many-to-one classification / regression
    - Sentiment classification, document classification ...
    - Word / Sentence embeddings for downstream tasks (e.g. recommender system)
  - Encoder-Decoder (e.g BART)
  - Decoder-only (e.g GPT)

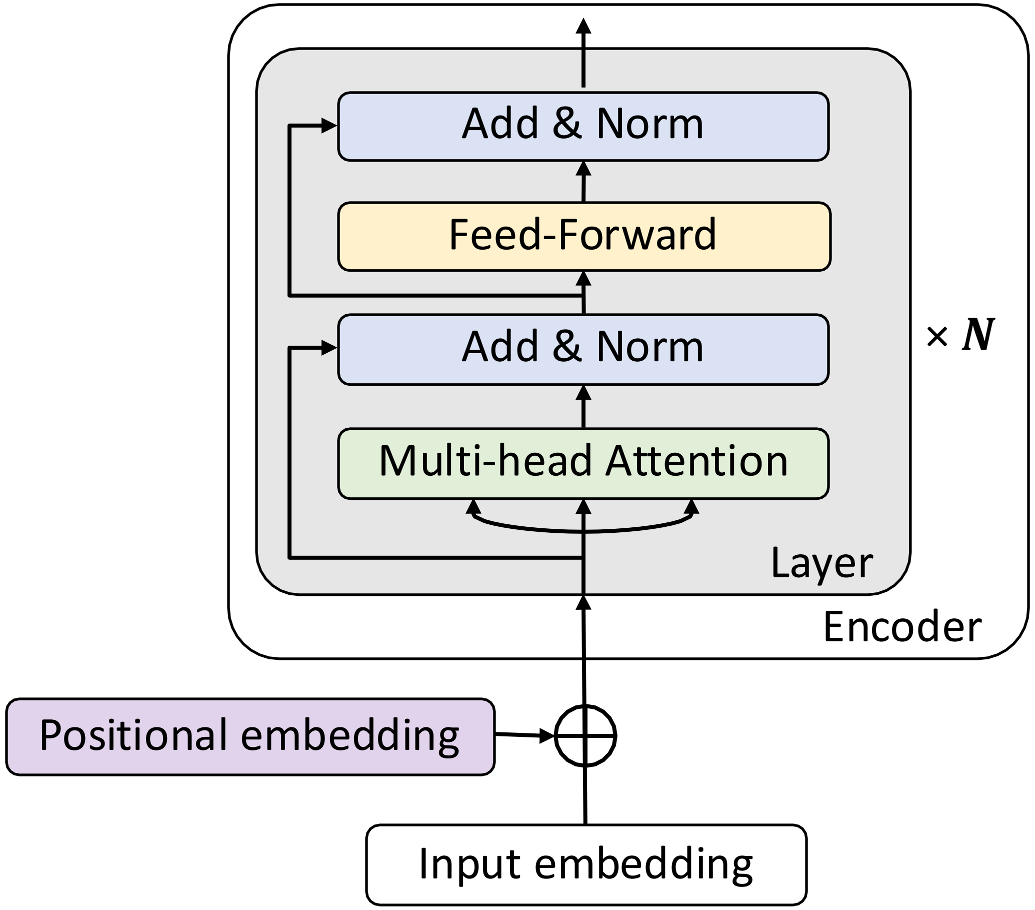


Transformers: Overview

- **Model usage**: Pre-training followed by fine- tuning. The transfered model can be:
  - Encoder-only (e.g BERT)
  - Encoder-Decoder (e.g BART)
    - Many-to-many use cases
    - Summarization, translation, style transfer ...
  - Decoder-only (e.g OpenAI GPT)
    - One-to-many use cases
    - Image / text / code generation, dialogue systems ...
    - GPT-3/4 based apps



Transformers: Overview

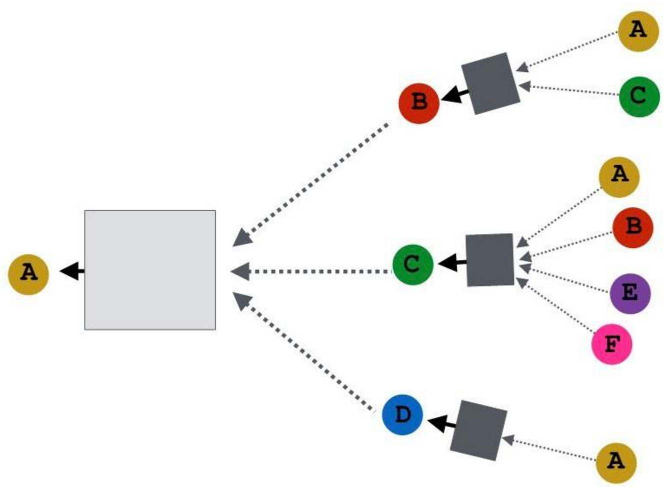


Design choices of transformers: (there are many papers on this topic for those interested in transformer architectures)

- Absolute/relative position, equivariant embedding (for graph)
- Sparse attention, low-rank attention, attention with prior, KV cache compression…
- Placement, substitutes, normalization-free
- Cross-block connections, recurrence/hierarchy, other architecture

Recap: Graph Attention Mechanism

- Message Aggregation: Review



Non-linearity

Embedding of u at layer l

$$\boldsymbol{h}_v^{(l)} = \sigma\Big(\sum_{u \in N(v)} \alpha_{vu} \boldsymbol{W}^{(l)} \boldsymbol{h}_u^{(l-1)}\Big)$$

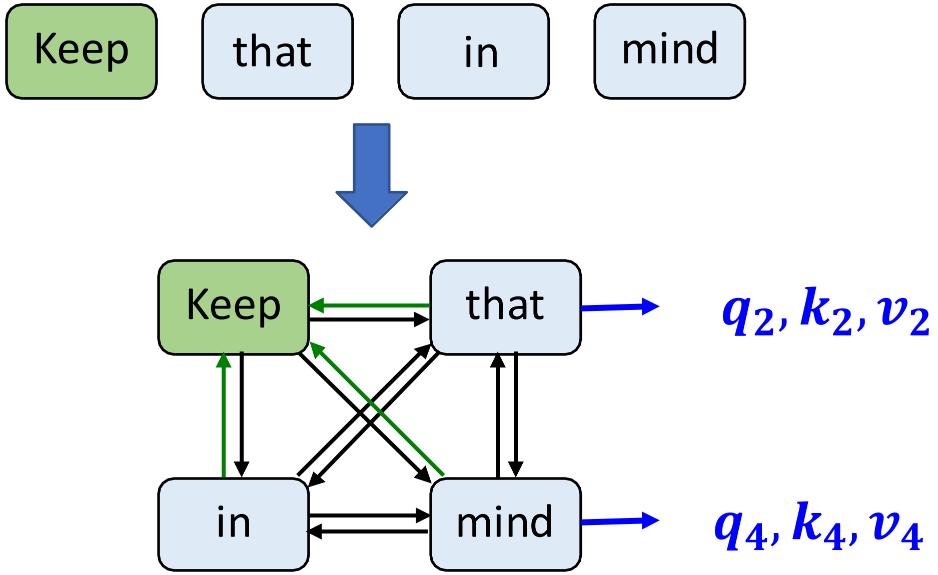
Neighbors of v

Learnable parameter

Weighting factor of u's message to v

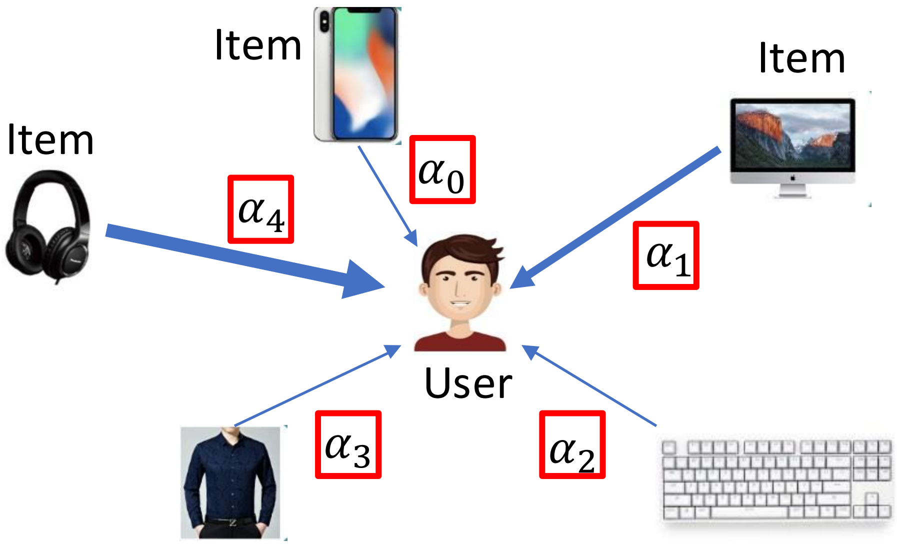
Transformers: in the Language of Graphs

Transformers



Step ① Mapping: Each node feature $\boldsymbol{x_i}$ is projected to $\boldsymbol{q_i}, \boldsymbol{k_i}, \boldsymbol{v_i}$.

GATs

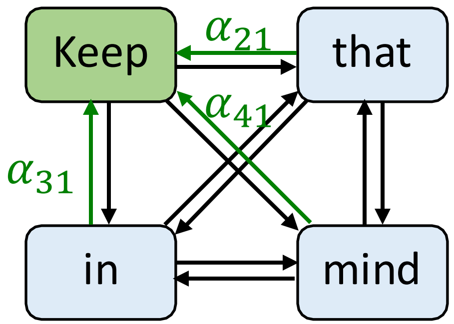


Message computing: transform information of neighbor node to a message.

$$m_u^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{h}_u^{(l-1)}, u \in N_v$$

Transformers: in the Language of Graphs

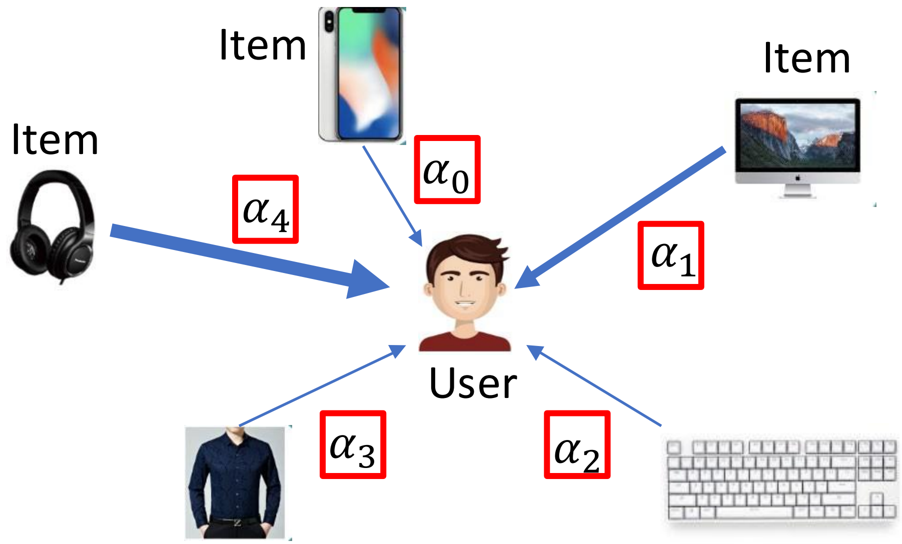
Transformers



Step ② Attention: Calculate the edge weights using q_i, k_j of the two endpoints node i and node j as $e_{ij} = \frac{q_i^T k_j}{\sqrt{d}}$, then normalizing it by neighbors of node i,

$$\alpha_{ij} = \mathrm{softmax}_i(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})}$$

GATs

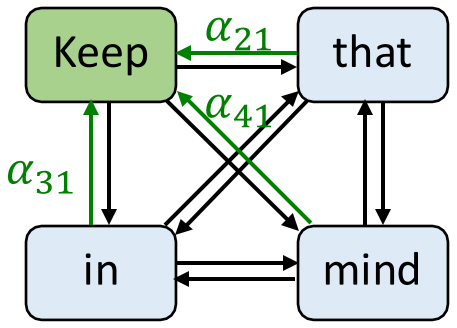


Attention computation: calculate the importance of neighbors

$$\alpha_{vu} = att(\boldsymbol{h}_v^{(l-1)}, \boldsymbol{h}_u^{(l-1)})$$

Transformers: in the Language of Graphs

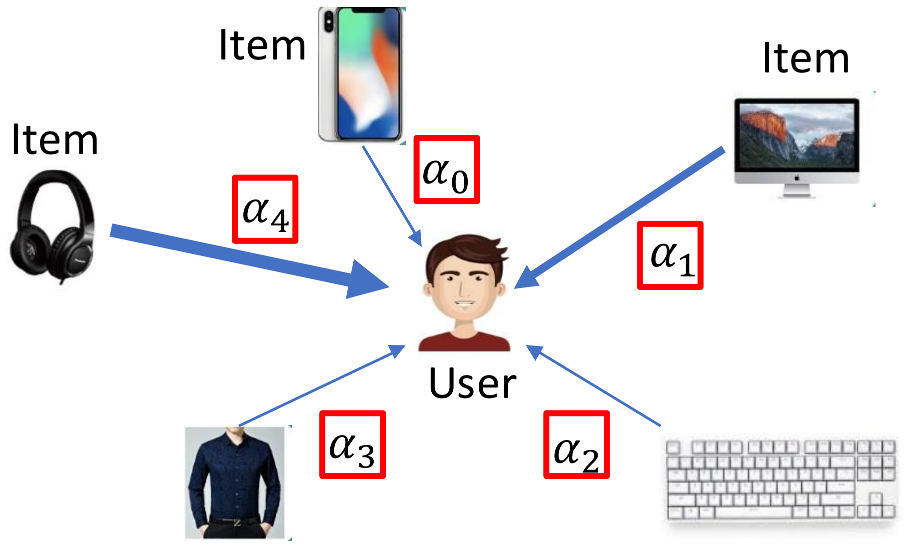
Transformers



Step ③ Update: Update each node feature according to its neighbors as

$$\boldsymbol{x}_i' = \sum_{\boldsymbol{k} \in \boldsymbol{N}_i} \alpha_{ij} \boldsymbol{x}_j$$

GATs



Aggregate message: aggregate messages from neighbor nodes

$$\boldsymbol{h}_v^{(l)} = \sigma\Big(\sum_{u \in N(v)} \alpha_{vu} \boldsymbol{m}_u^{(l)}\Big)$$

Transformers: in the Language of Graphs

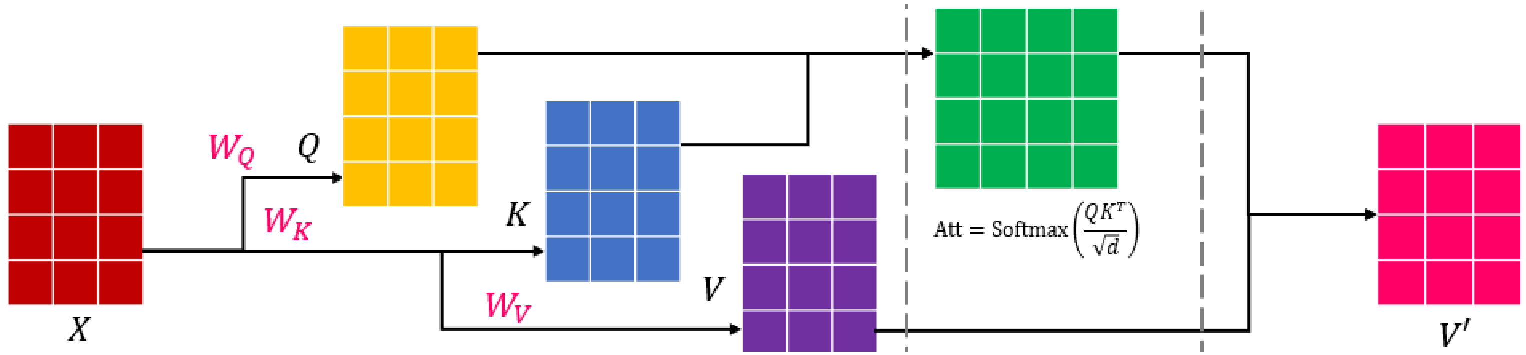
- Summary: Comparison of **Self-attention (SA)** and **Graph Attention Networks (GAT)**
  - Step ① Mapping
    - **SA**: **different** weights for q, k, v. $q = w_q x, k = w_k x, v = w_v x$.
    - **GAT**: **shared** weights for q, k, v. $q = wx, k = wx, v = wx$.
  - Step ② Attention: **SA** uses dot-product attention, while (the original) **GAT** uses concatenation with MLP
    - Dot-product: $e_{ij} = \frac{q_i^T k_j}{\sqrt{d}}$.
    - Concat: $e_{ij} = \mathrm{act}(W[q_i \parallel k_j])$, where is a weight vector and act is the activation function like $\mathrm{LeakyReLU}$

Transformers: in the Language of Graphs

- The above computations do not require the assumption of the **complete graph**.
  - We assume full connectivity, mostly because we do not want to miss any potential token correlations.
- **Self-attention can be easily adapted to graph-structured** input data where the token correlations are given by the **adjacency matrix**, by replacing the **complete graph** with the **input graph**.
  - $\mathrm{Self{-}Att}(X) = \mathrm{Softmax}(\frac{(\boldsymbol{W}_k\boldsymbol{X})(\boldsymbol{W}_q\boldsymbol{X})^T)}{\sqrt{d}} \odot \boldsymbol{A_G} \odot \boldsymbol{W}_E E)V$.
  - A_G is the adjacency matrix of graph E is the edge weights of the graph is any.
- The complexity is no longer $O(n^2 d)$ but is linear to the edge number $O(E)$.

Sparse Transformers for Efficiency

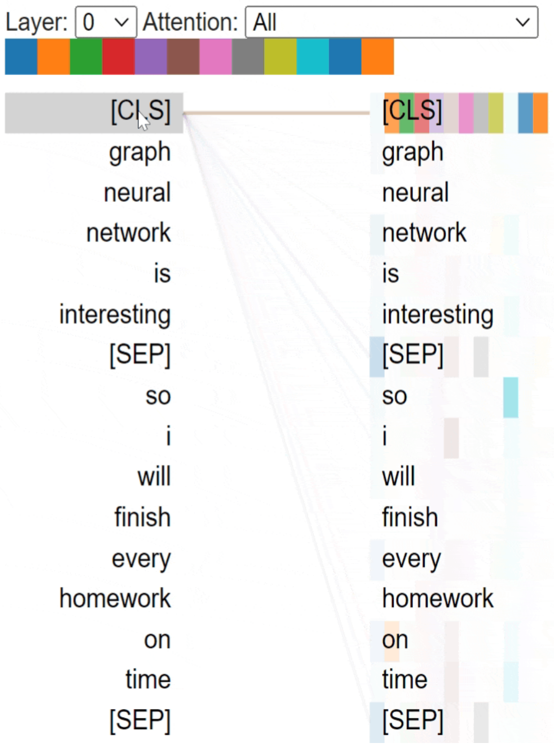
- Conventional Transformers cannot scale to **long sequences** due to $\boldsymbol{O(n^2)}$ complexity from the full-attention
  - The $\boldsymbol{QK^T}$ matrix multiplication, $\mathrm{Softmax}()$, $\mathrm{Att}V$ value updates all consume n^2 time and memory.



Complexity: $O(ndd_X)$ $\quad O(n^2d)$ $\quad O(n^2d)$

Sparse Transformers for Efficiency

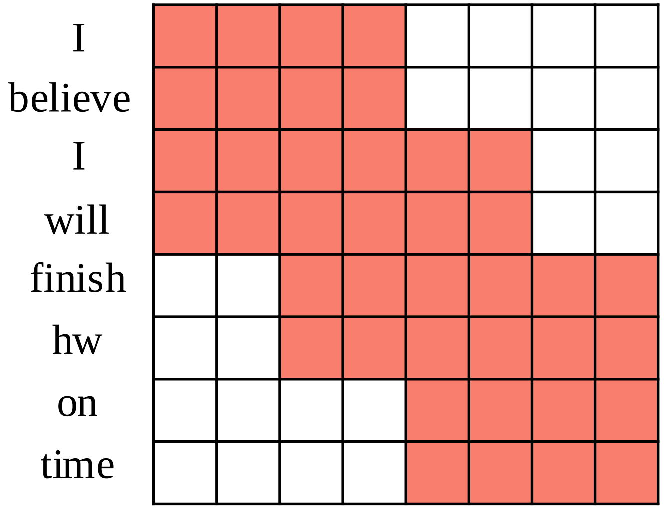
- **Observation 1**: Although every attention is calculated, most of them are close to 0, the resulting attention maps are usually **sparse**.
- **Observation 2**: non-zero attention mostly appear between the node and its local neighbors. (**local** attention).
- **Observation 3**: some key words like "so" almost attend to every token in the sentence. (**global** attention)
- **Can we simplify self-attention (full-attention) using graph?**



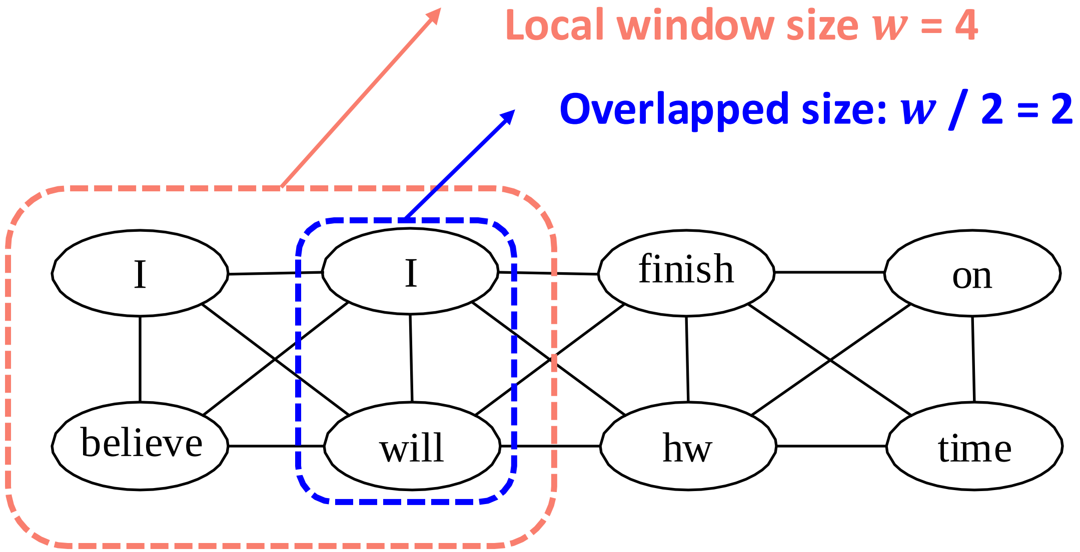
Try yourself!
https://github.com/jessevig/bertviz

Sparse Transformers: Longformer

- Applying overlapped local window attention to approximate the full-attention, only calculating attentions shaded in red



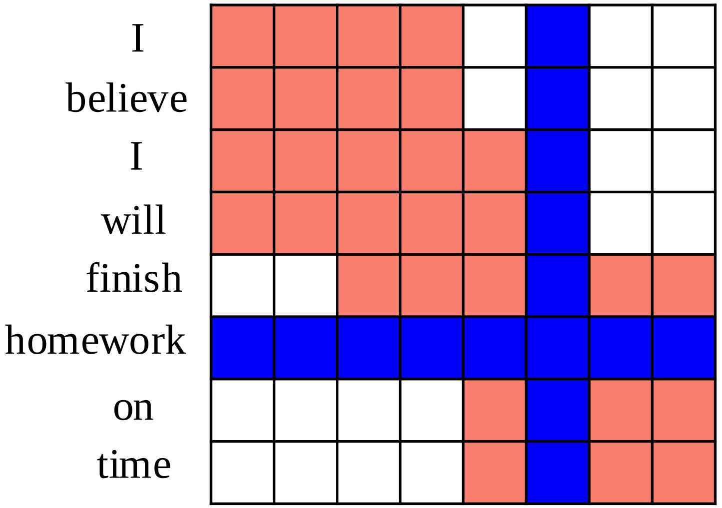
Masked Attention Pattern
(adjacency matrix)



Associated graph structure

Sparse Transformers: Longformer

- Longformer is based on the assumption that adjacent words have stronger correlations

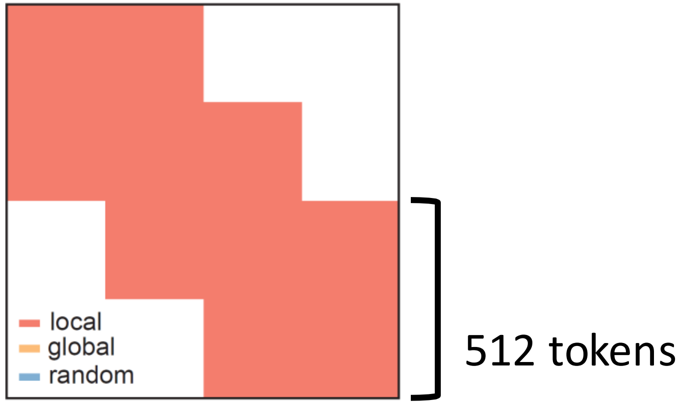


Masked Attention Pattern
(adjacency matrix)

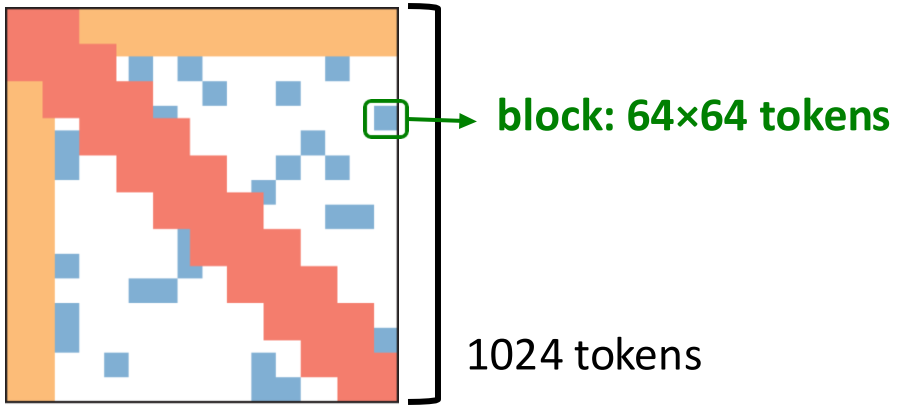
- **Local window** is overlapped in half to enable cross- window attention (to ensure the graph is connected so that every pair of tokens can attend by stacking layers).
- **Global attention** is further introduced for specific down-stream task
  - Select a subset of random tokens as global tokens
  - Use special token such as beginning-of-sentence token
- Complexity is now $O(nw)$ compared to $O(n^2)$.
- Longformer can handle long sequences like 4096 tokens, by specifying local window size to be 512

Sparse Transformers — BigBird

- BigBird model further introduces **Random Attention** to better approximate the full- attention.
- The smallest unit in BigBird is called a **block** (64 adjacent tokens)
- "Blockifying" is used to accelerate the sparse attention computation



(a) Longformer



(b) BigBird

Beyond Sparse Graphs: Graph Transformers

Graph Transformers

Graph Transformers: Overview

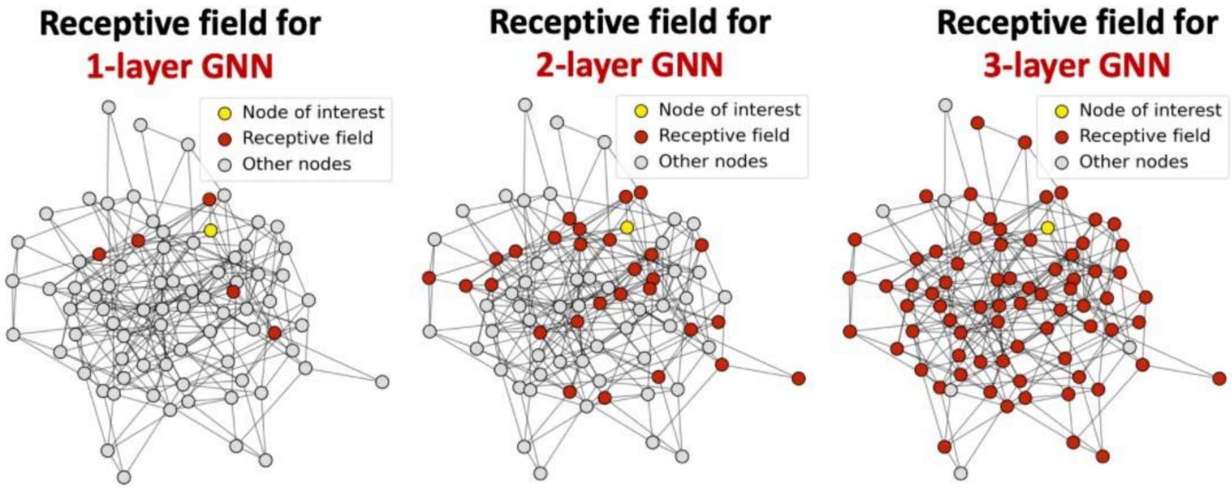
- Motivation of Graph Transformers
- Challenges for Building Graph Transformers
- Encodings: Positional & Structural
  - Laplacian Positional Encoding
  - Random Walk Structural Encoding
- Token Construction
- Forward Propagation
- Empirical Verification

Message Passing Drawbacks

- **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

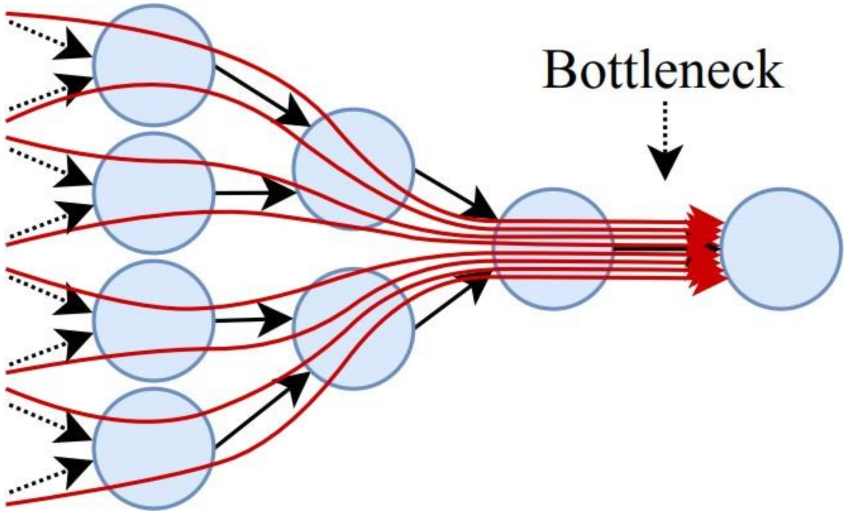
- **Over-smoothing:** node features tend to converge to the same value with the increasing number of message passing layers



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

Message Passing Drawbacks

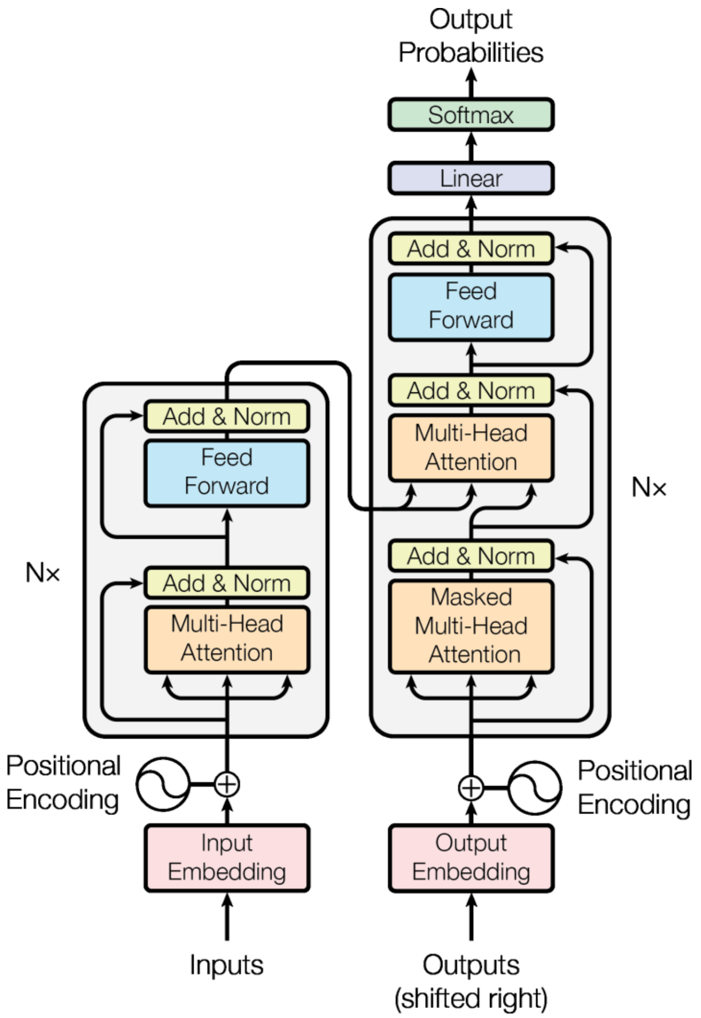
- **Over-squashing**: In tasks relying on long-distance interactions, an **exponentially-growing** amount of information from distant nodes is squashed into a fixed-size vector.
- Message Passing Layers cannot **capture long-range dependencies effectively**.



The bottleneck of graph neural networks

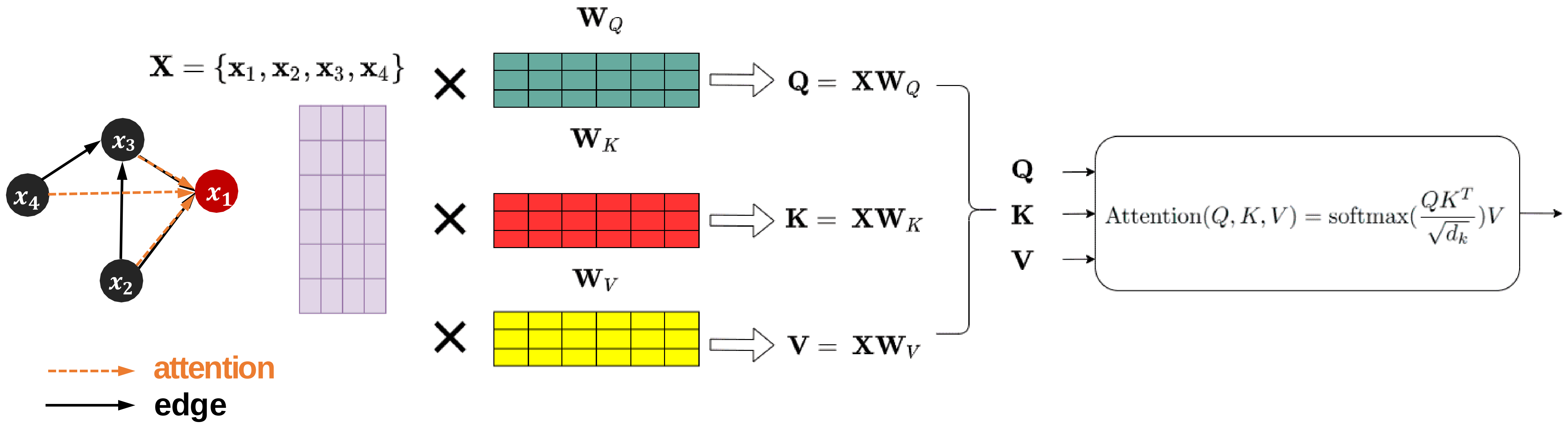
Motivation of Graph Transformer

- **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

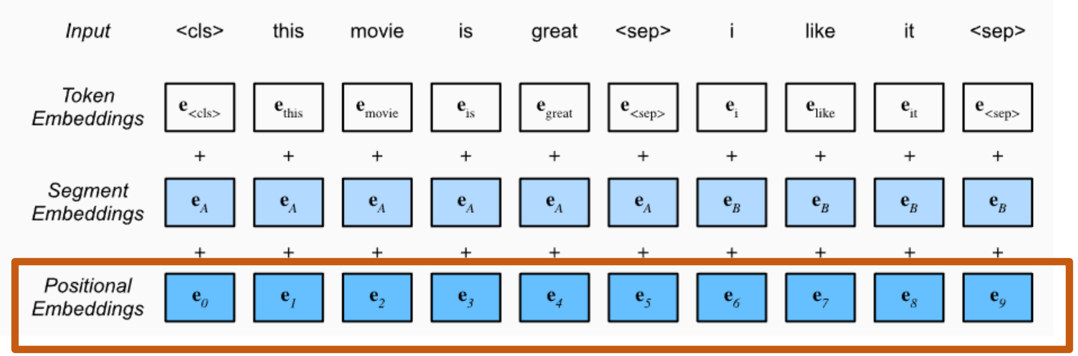
- Naïve full attention on node set:

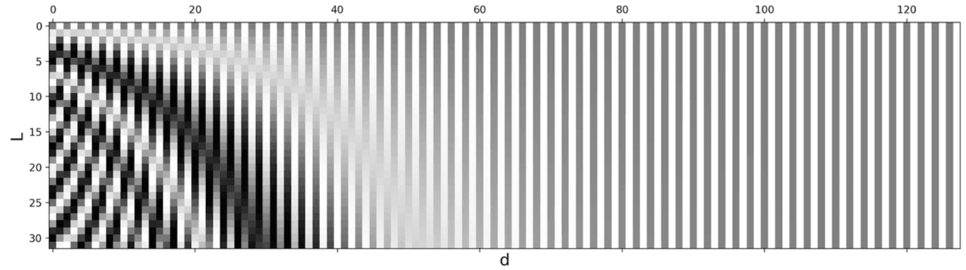


- Naturally capture **long-range dependencies**

Challenges for Graph Transformers

- Language has natural sequential order, while graphs are permutation invariant to node ordering.





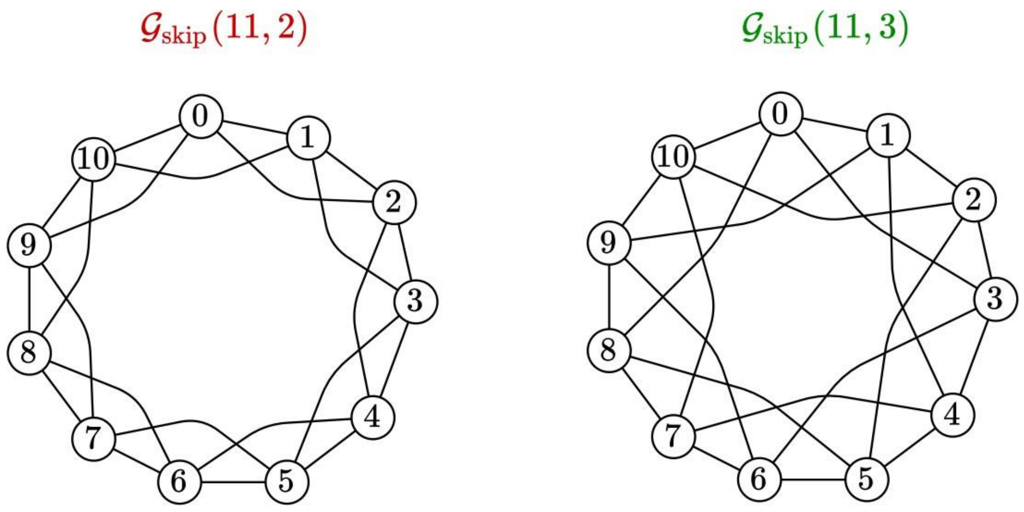
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right),$$

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

- 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

- 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)**

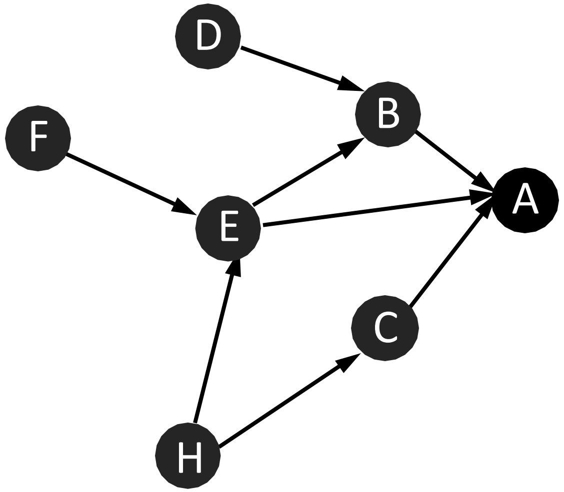


Circular Skip Links (CSL) Graphs

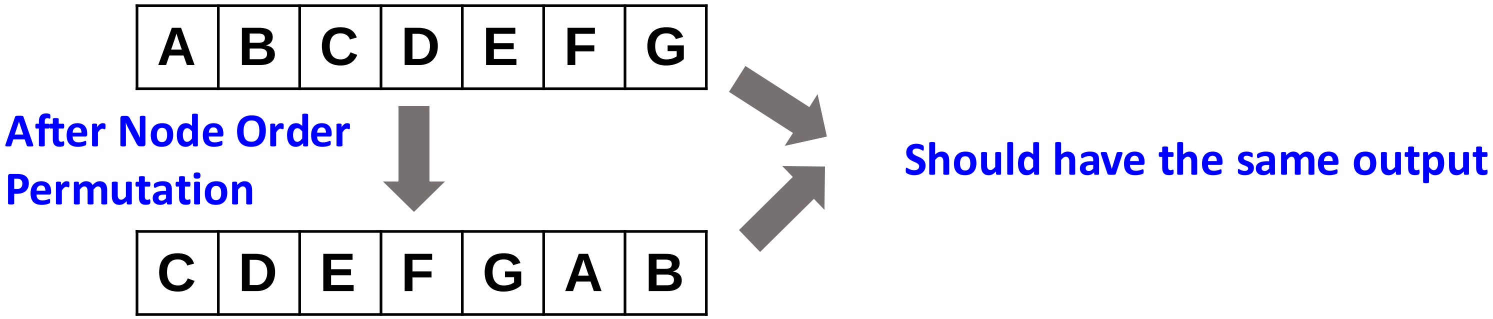
- Message passing GNNs fail to distinguish Circular Skip Links (CSL) graph pair
- Structural Encoding (which captures local substructure) can distinguish graph pairs

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

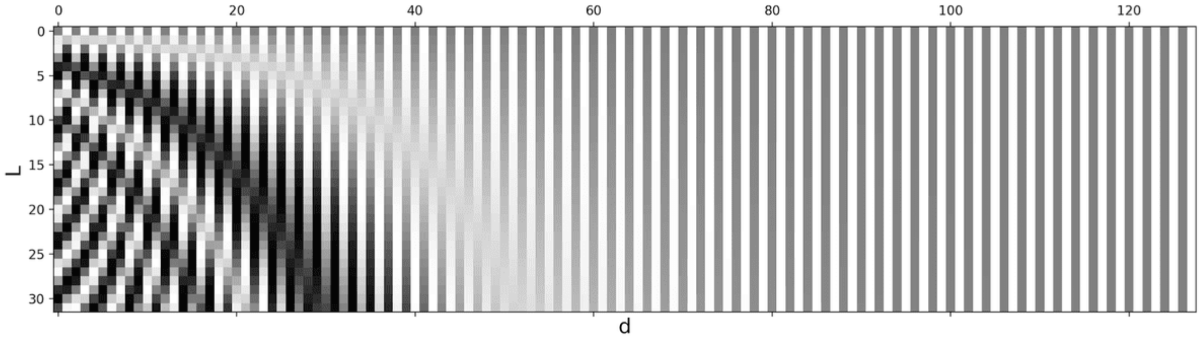


Differences between PE and SE

- **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.
  - Recall: Positional-aware GNN. In practice: pair-wise shortest path distance
- **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.
  - Recall: Identity-aware GNN. **Next:** Laplacian & Random walk

Why Sin/Cos as Positional Encoding

- The positional encoding for sequential input:



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

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

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

Definition of Laplacian Operator:

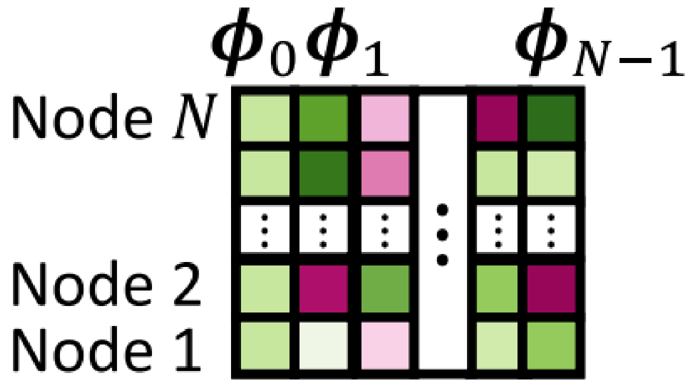
$$Lf = \mathrm{div}(\nabla f)$$

Laplacian in Graph Domain

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

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

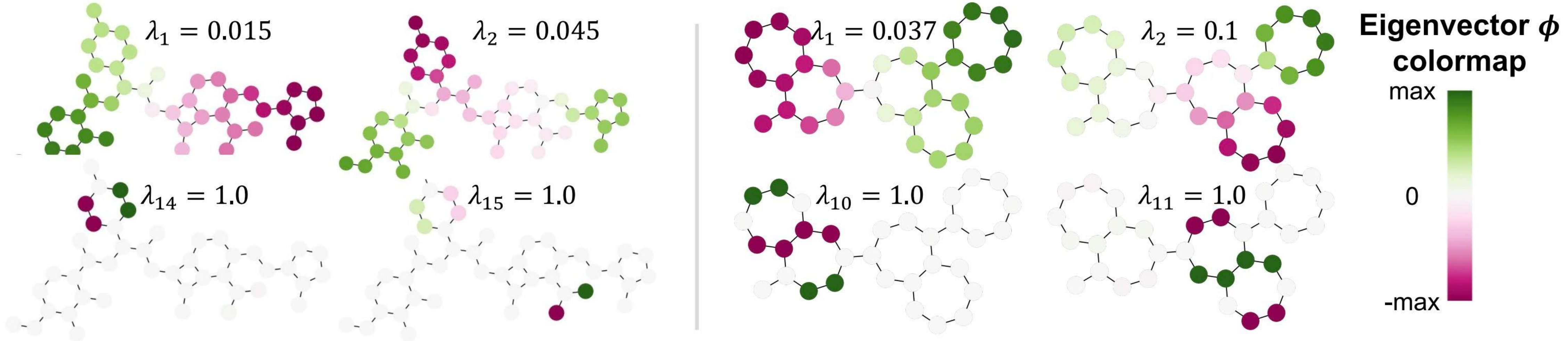
- $U = [\phi_0, \ldots, \phi_{N-1}]^T$, where ϕ_i indicates the i-th eigenvector.



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



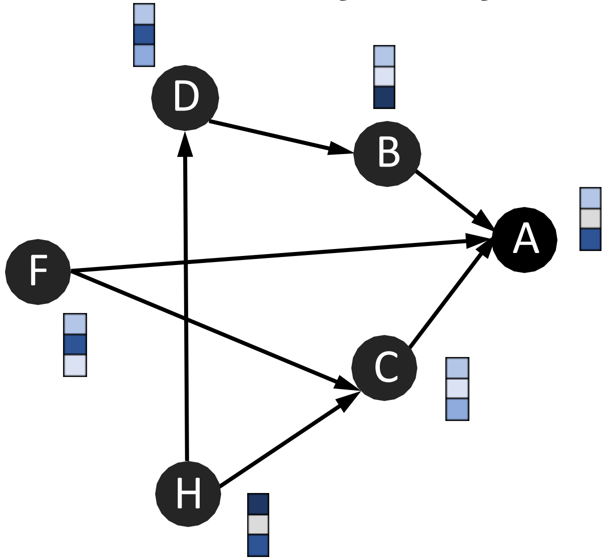
The **low-frequency** eigenvectors ϕ_1, ϕ_2 are **spread across the graph**

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

Laplacian Positional Encoding

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

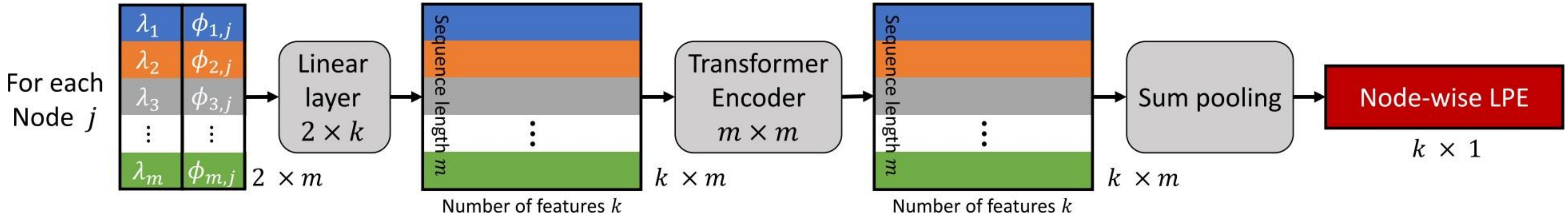
$$p_j^{\text{LapPE}} = [\phi_{1j}, \phi_{2j}, \dots, \phi_{mj}]$$



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

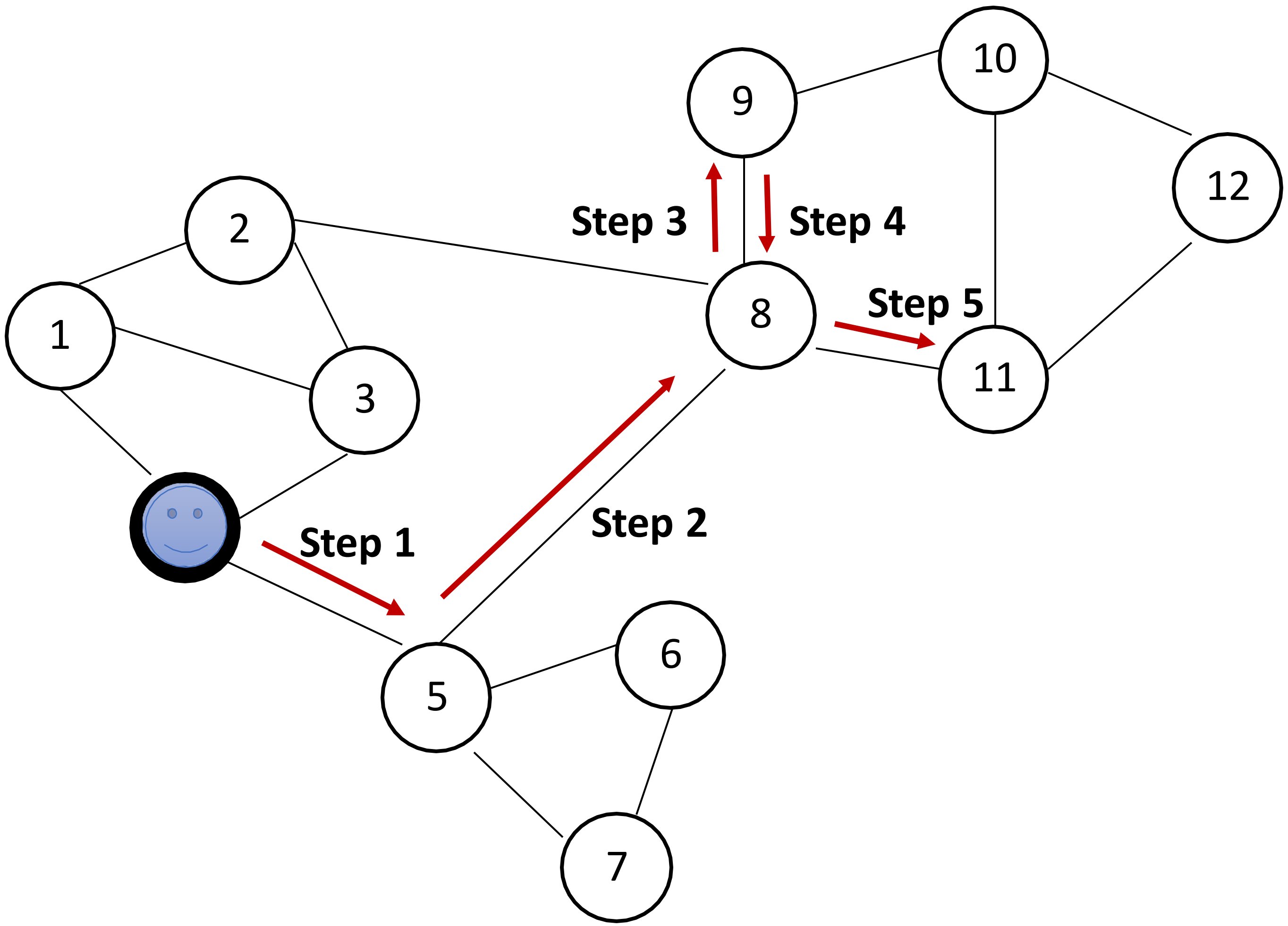
Laplacian Positional Encoding

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



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

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

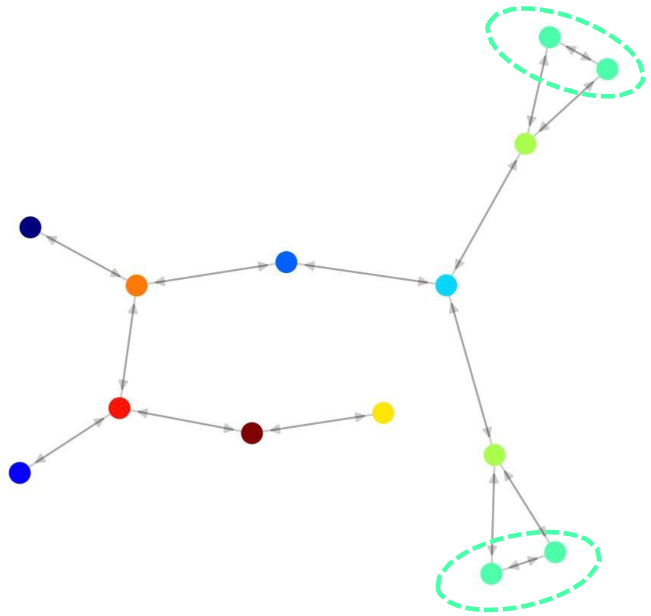
- $\mathrm{RW} = AD^{-1}$ is the **random walk operator**. A is the adjacency matrix, D is the degree matrix.
- **Random Walk Encoding (RWE)** for the i-th node is defined with k-steps of random walk as

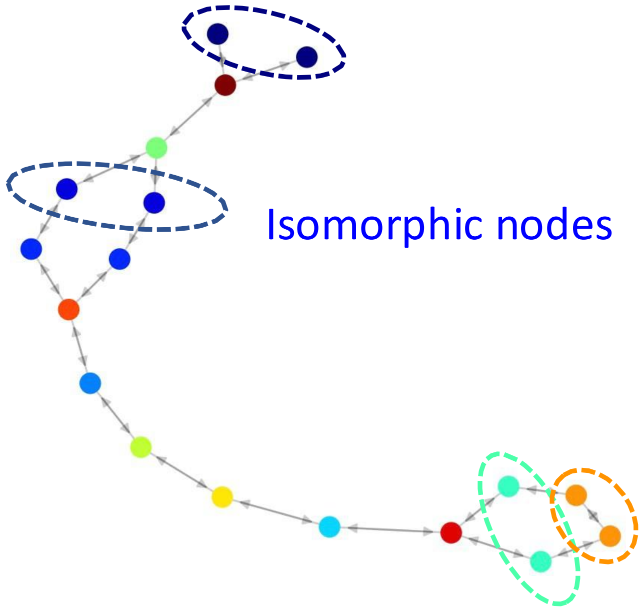
$$p_i^{RWE} = \left[RW_{ii}, RW_{ii}^2, \dots, RW_{ii}^k\right] \in \mathbb{R}^k$$

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

- Question:
  - What happens when $\boldsymbol{k}$ increases? (Higher-order neighborhood is considered)
  - What happens to the RWE if a node is densely connected to its neighborhood?

Random Walk Encoding Visualization

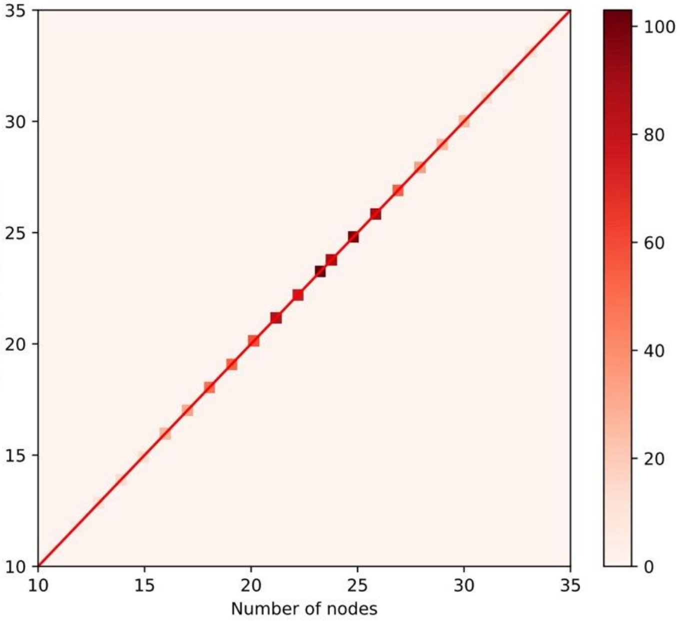




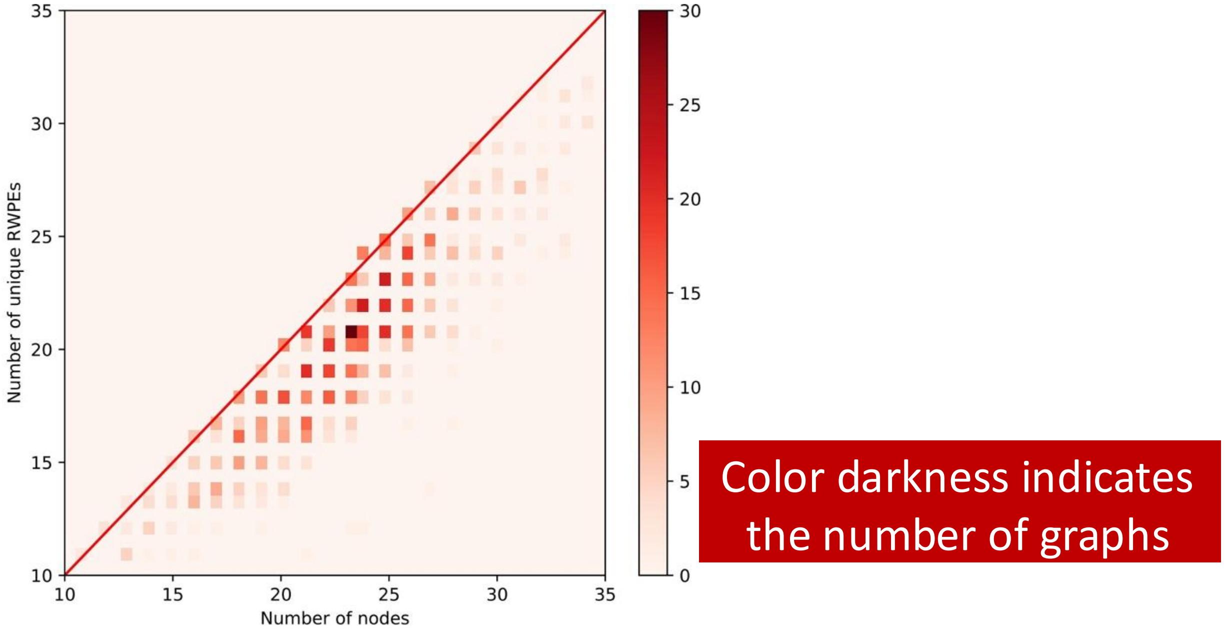
- **Different node color** represents a **unique RWE** vector with $k = 24$
- The nodes with the same color / RWE are isomorphic in the graph, i.e. their k-hop structural neighborhoods are the same!

Comparison between RW and Laplacian

- LapPE and RWE on ZINC validation set:



(a) LapPE, $k = 36$

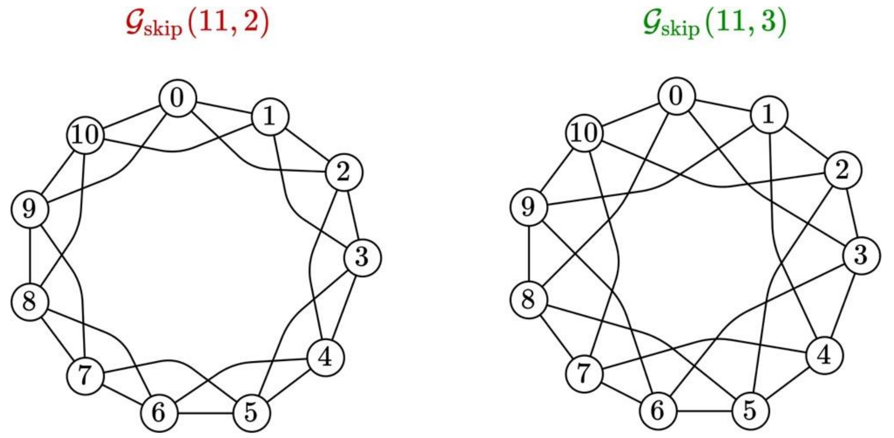


(b) RWPE, $k = 24$

- Laplacian PE guarantees unique node representations better (an injective mapping)
- With RWE, most graphs have a large number of nodes with unique positional encoding

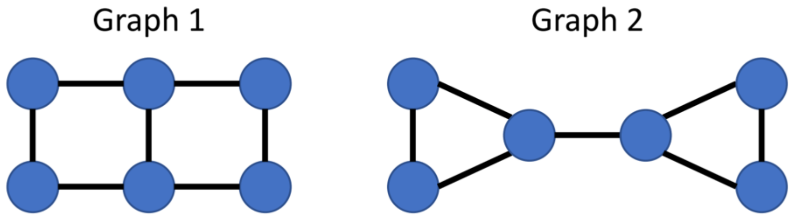
Laplacian & RW Improves Expressiveness

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



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

4-dimensional RWE distinguish CSL graph pairs

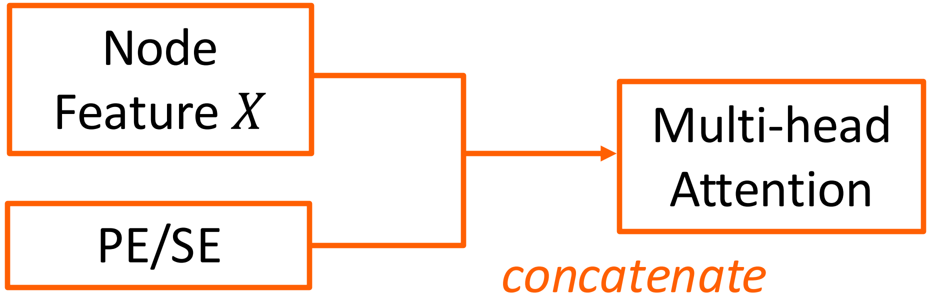


Eigenvalues of Graph 1	Eigenvalues of Graph 2
0	0
1	0.438
2	3
3	3
3	3
5	4.562

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

How to inject PE/SE

- **Concatenate with node/edge/graph** features before Attention layers
  - Example: SAN, GPS. Note: add may lead to dimension mismatch



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

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

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

How to inject PE/SE

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

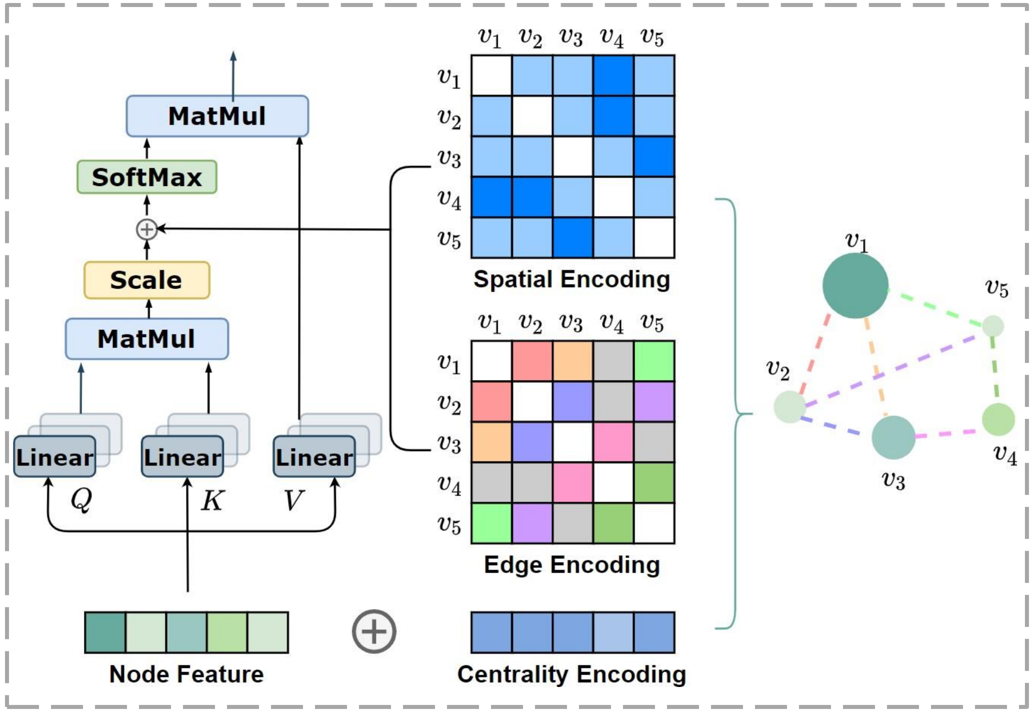
Attention: $e_{ij} = \frac{(\boldsymbol{h}_i \boldsymbol{W}_q)(\boldsymbol{h}_j \boldsymbol{W}_k)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)} + c_{ij}$

Spatial Encoding:

Shortest path between v_i, v_j

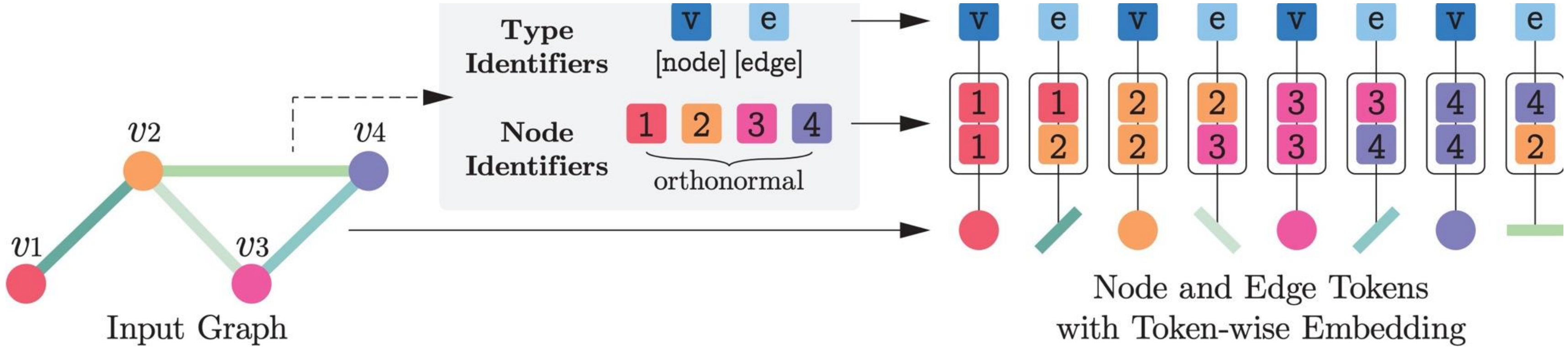
Edge Encoding:

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



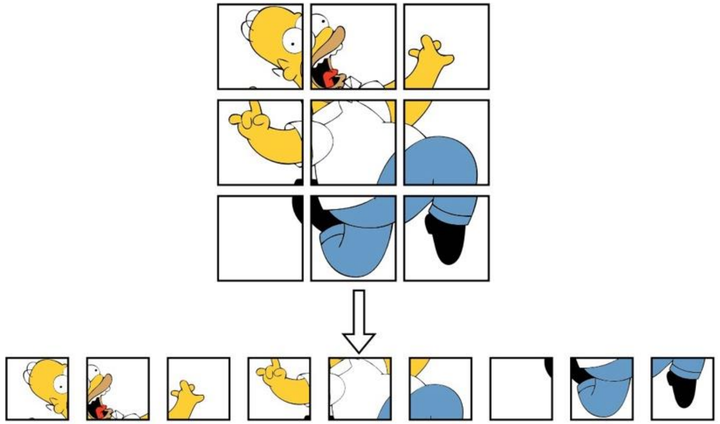
How about Tokens in Transformer?

- 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 $O(N+E)^2$

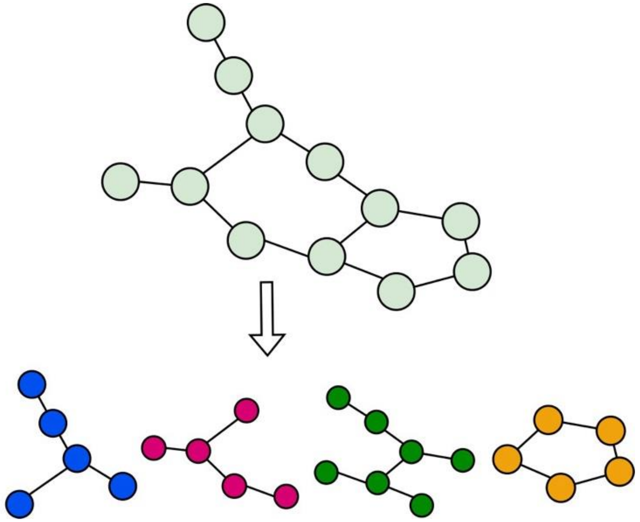


Token Construction

- 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



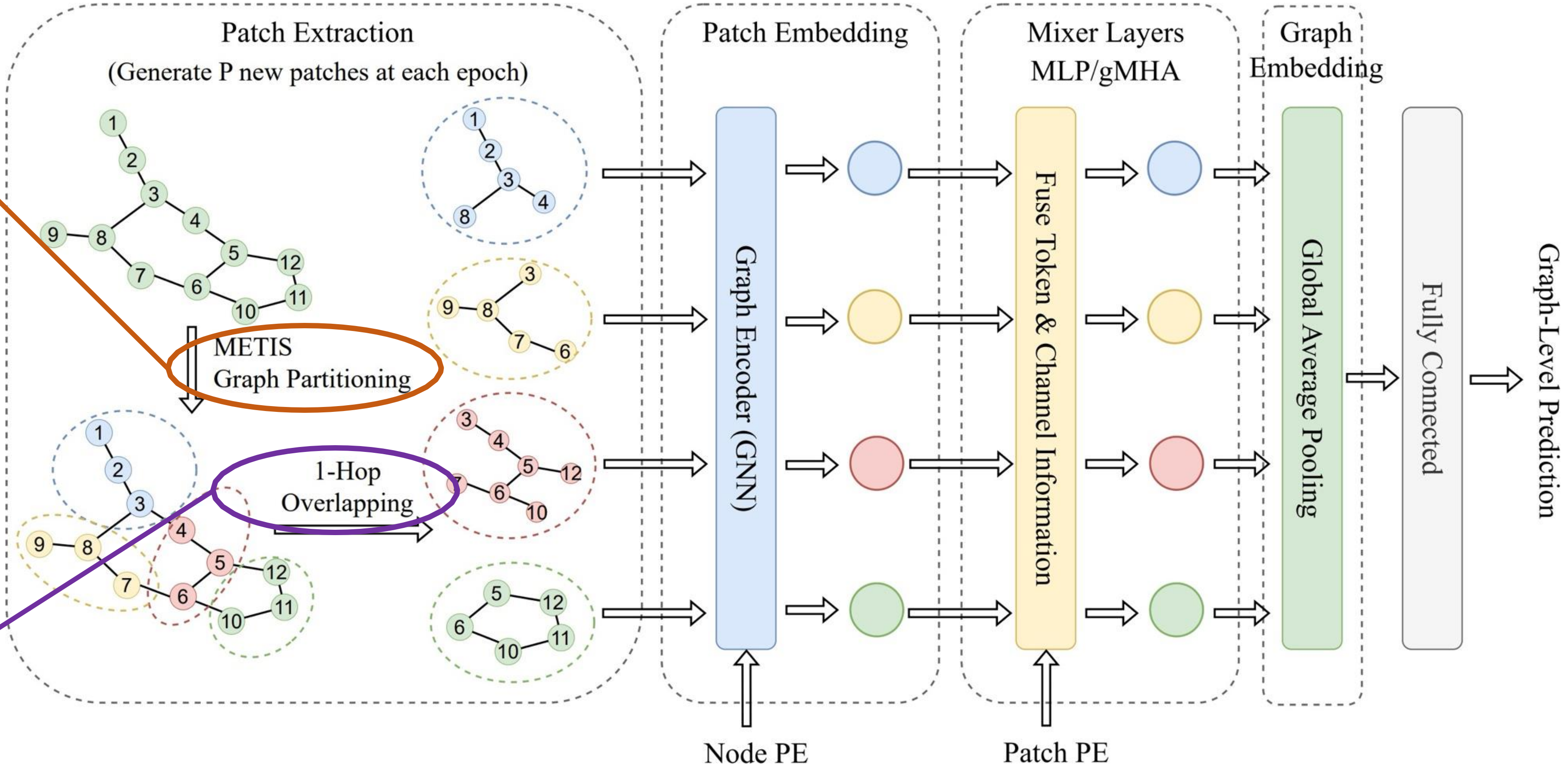
Images



Graphs

MLP-Mixer Architecture

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



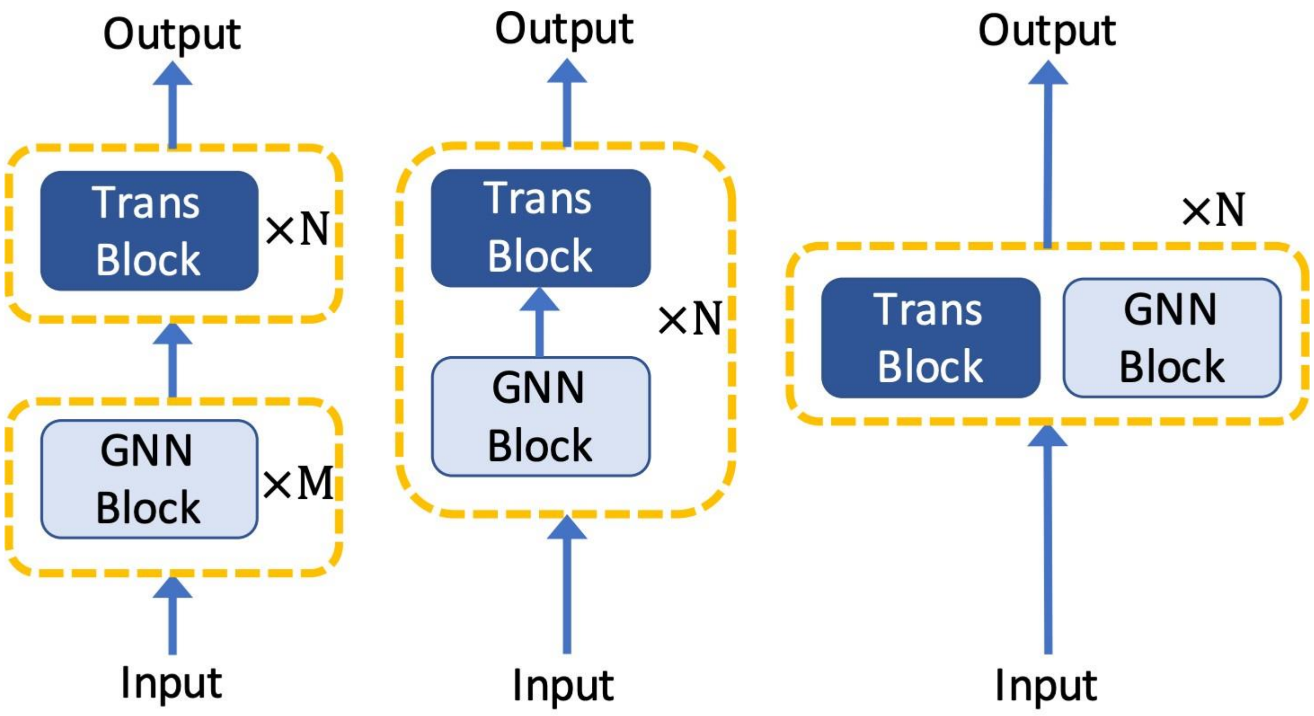
- Involve all 1-hop neighbors to capture important edge information
- e.g., the cutting edges

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

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

Forward Propagation

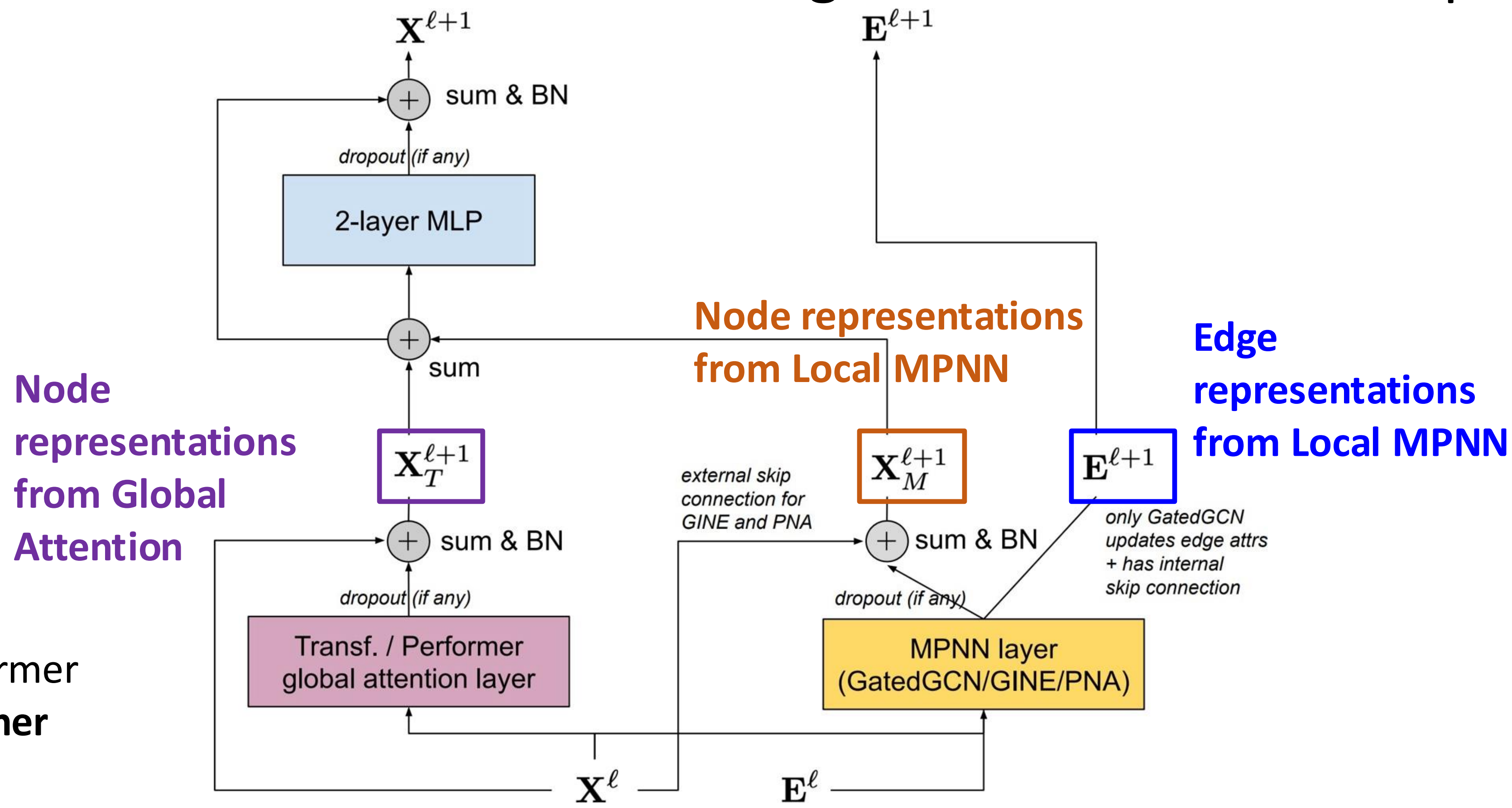
- GNNs can be used as **auxiliary modules with transformer** architectures



(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

Empirical Verification (Expressiveness)

Model	Easy EDGES	Medium TRIANGLES-SMALL	 TRIANGLES-LARGE	Hard 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:

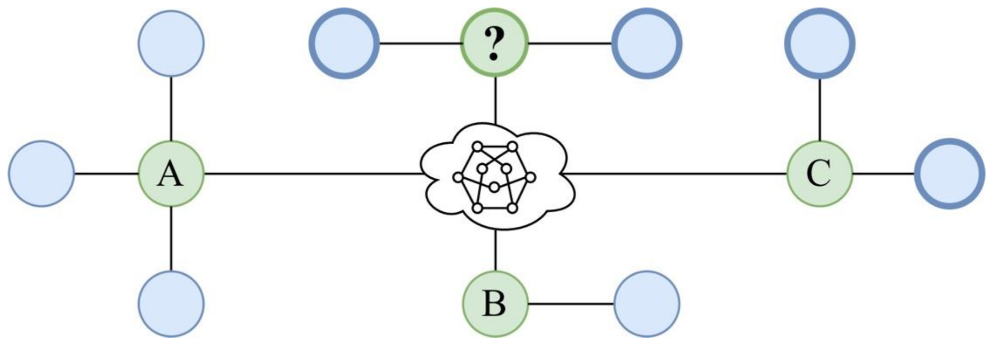
Model (PE/SE type)	ACTOR	CORNELL	TEXAS	WISCONSIN	CHAMELEON	SQUIRREL
Geom-GCN [Pei *et al.*, 2020]	31.59 ±1.15	60.54 ±3.67	64.51 ±3.66	66.76 ±2.72	**60.00 ±2.81**	38.15 ±0.92
GCN (no PE/SE)	33.92 ±0.63	53.78 ±3.07	65.95 ±3.67	66.67 ±2.63	43.14 ±1.33	30.70 ±1.17
GCN (LapPE)	34.30 ±1.12	56.22 ±2.65	65.95 ±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

- Geom-GCN is specialized for over-smoothing issue
- PE/SE have minimal effect on GCN's performance
- Global attention of a transformer empirically facilitates more successful information propagation

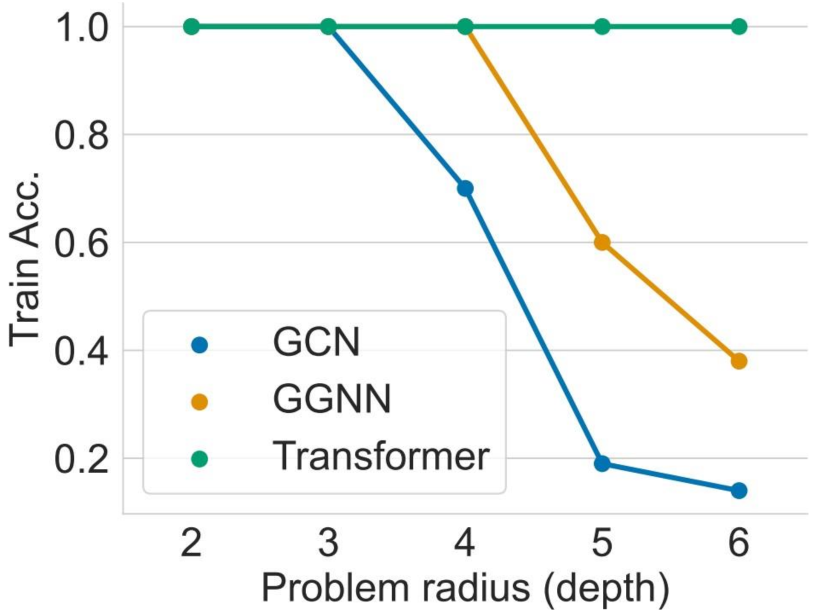
Empirical Verification (Long-range Dependencies)

- 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.