

# GNN, Graphformer, Body Transformer

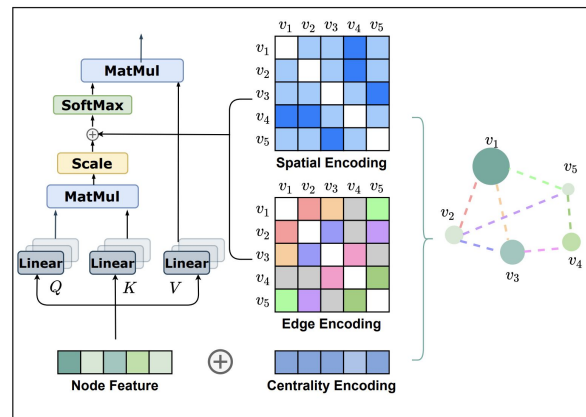
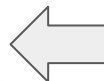
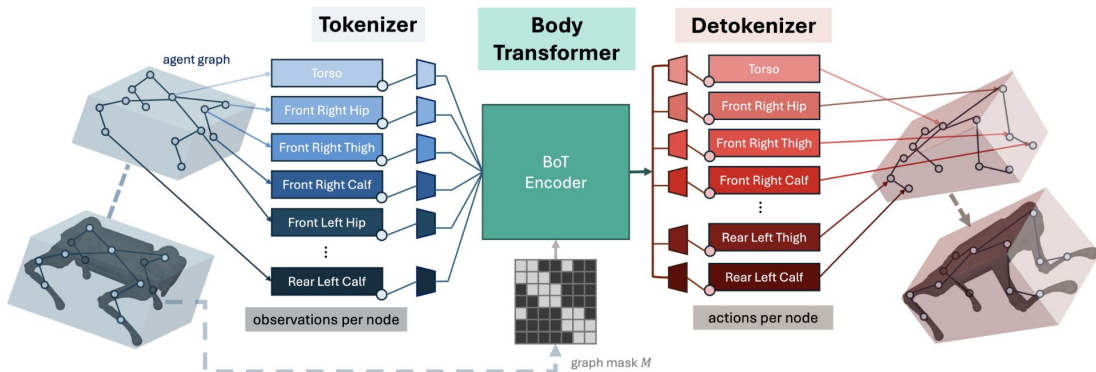
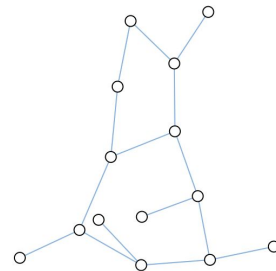
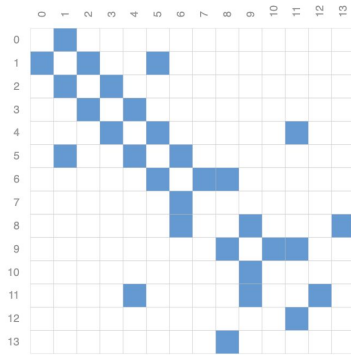
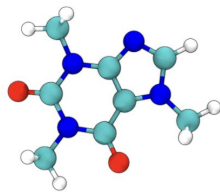
This is a kind reminder to the person presenting this presentation to hit the record button because he has forgotten to do it in every other MLP reading group presentation. Please please please if he is NOT recording, you have permission to turn on discord light mode on his laptop, block reddit, make him go to sleep before midnight, and make his caffeine intake go from >200mg a day to 0mg.

“Regardless of if he hits the record button, make sure to also install linux on his computer.” - person who nearly burned down their house

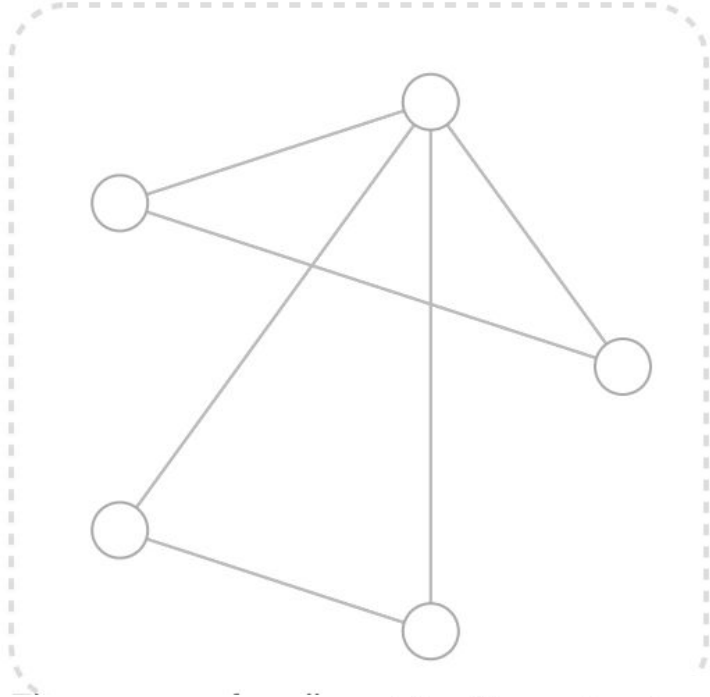
By: Alexiy Buynitsky

# Overview

1. GNNs
2. Graphformer
3. Body Transformer



# Graphs:



Undirected edge

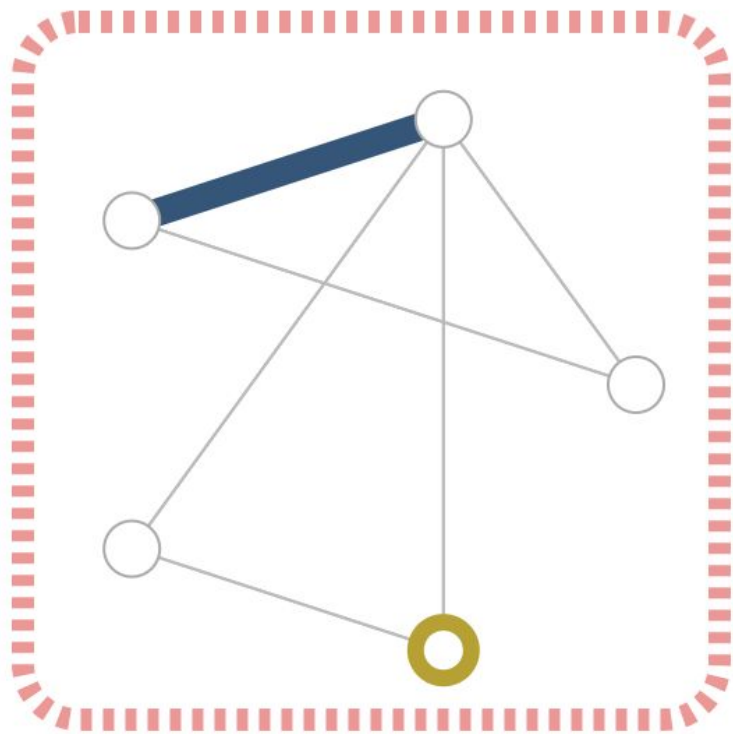


- V** Vertex (or node) attributes  
e.g., node identity, number of neighbors
- E** Edge (or link) attributes and directions  
e.g., edge identity, edge weight
- U** Global (or master node) attributes  
e.g., number of nodes, longest path

Directed edge



# Graphs:



Vertex (or node) embedding



Edge (or link) attributes and embedding



Global (or master node) embedding



# Graphs (image example):

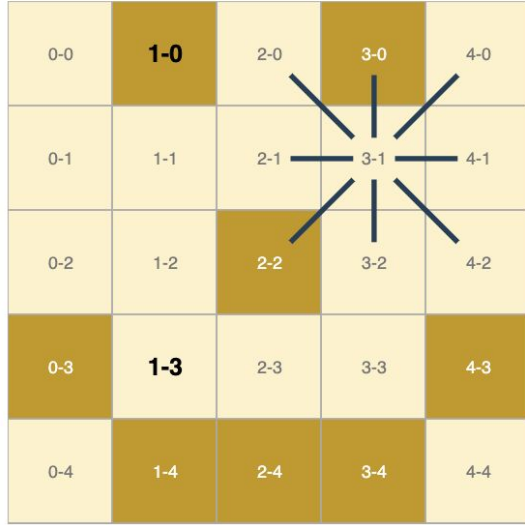
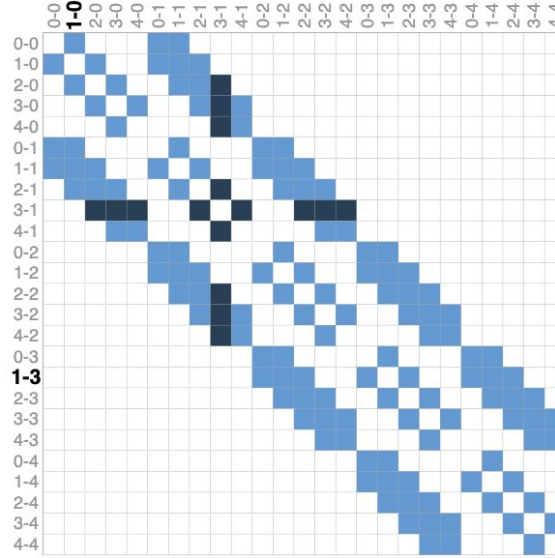
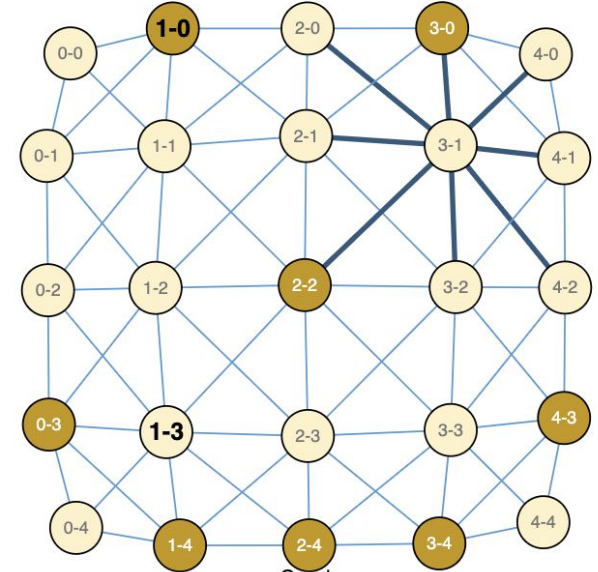


Image Pixels

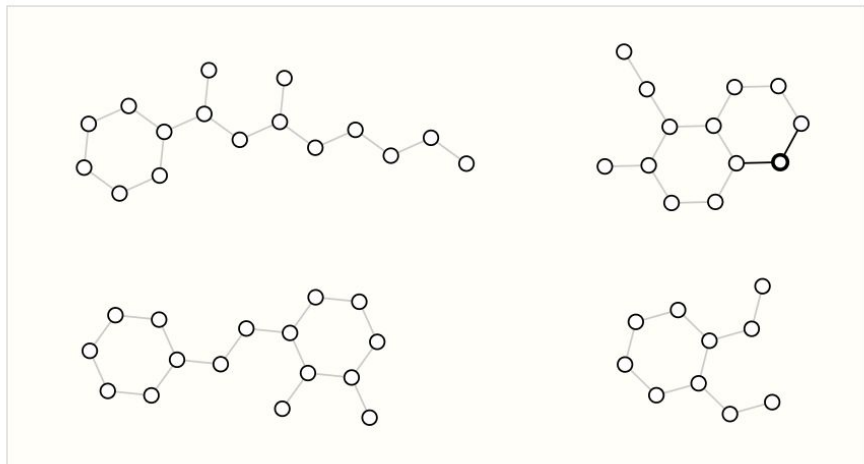


Adjacency Matrix

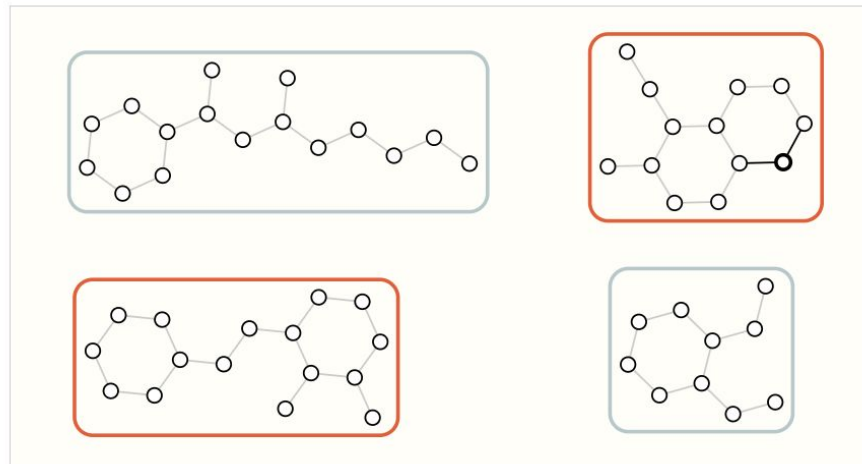


# Graph-level task:

- Predict properties of entire graph



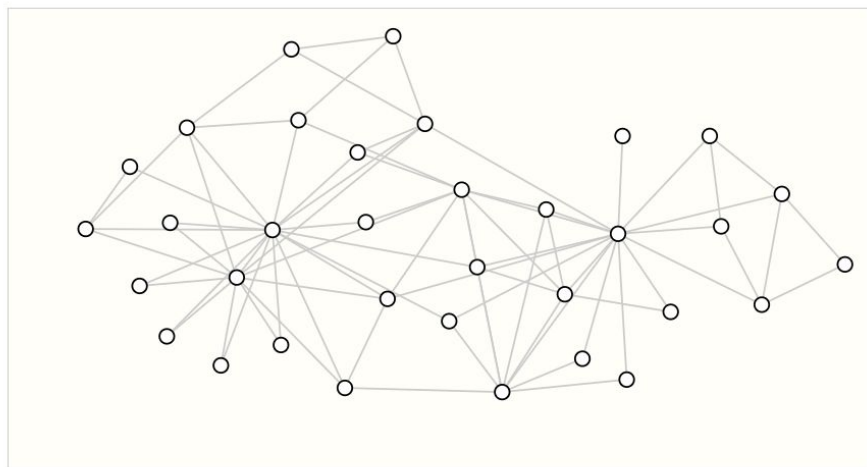
**Input:** graphs



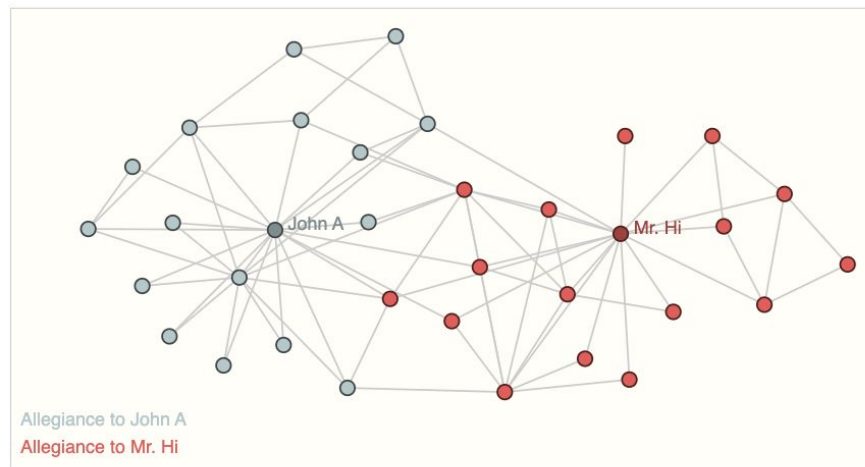
**Output:** labels for each graph, (e.g., "does the graph contain two rings?")

# Node-level task:

- Predict properties of each node



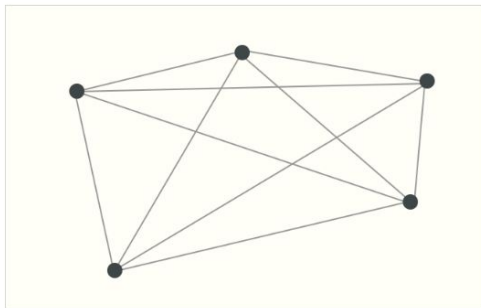
**Input:** graph with unlabeled nodes



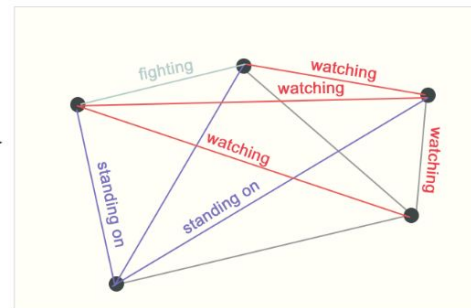
**Output:** graph node labels

# Edge-level task:

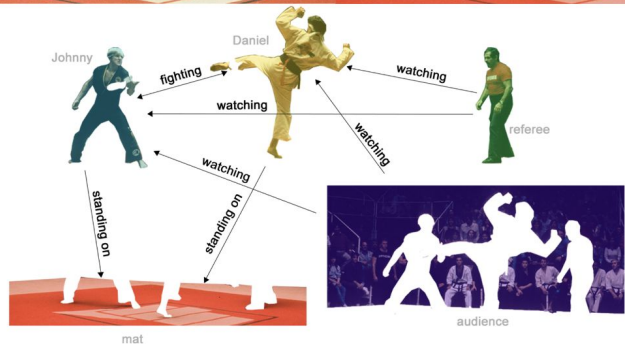
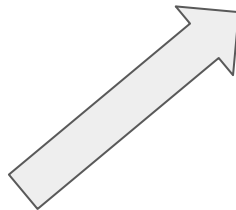
- Predict properties of each edge



Input: fully connected graph, unlabeled edges

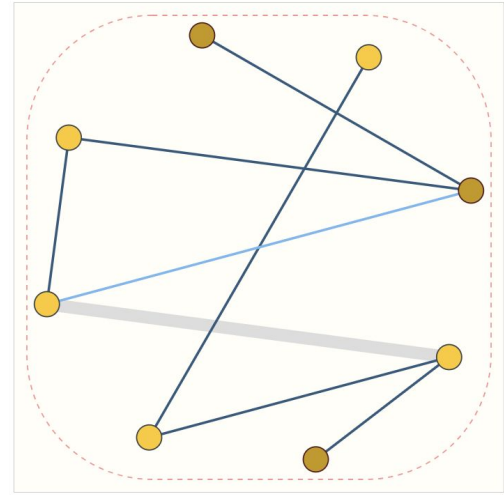
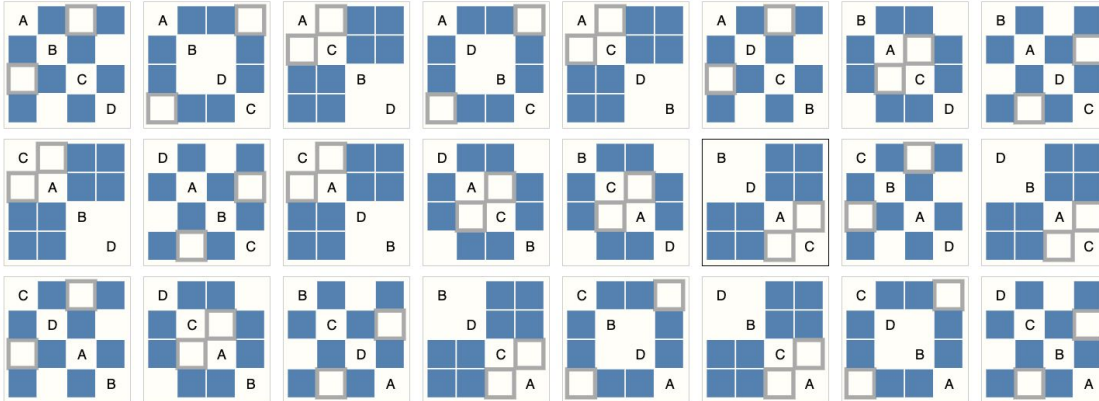
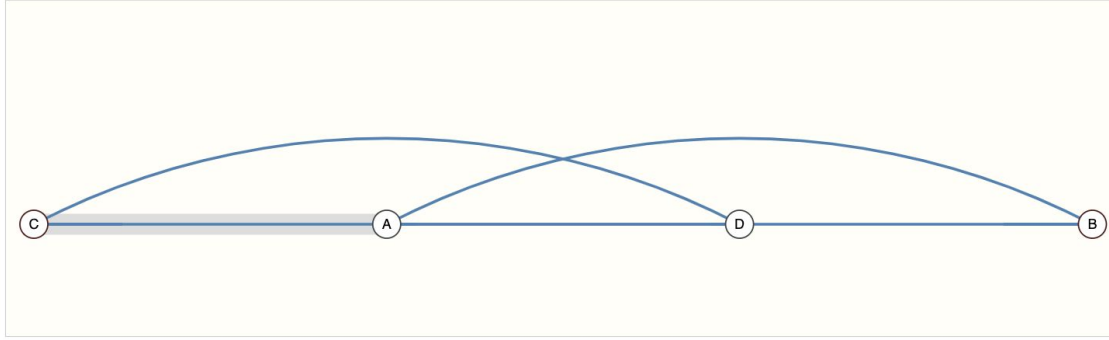


Output: labels for edges





# Permutation Invariance in graph:



Nodes

[0, 1, 0, 0, 0, 1, 0, 1]

Edges

[1, 1, 1, 1, 2, 1, 1]

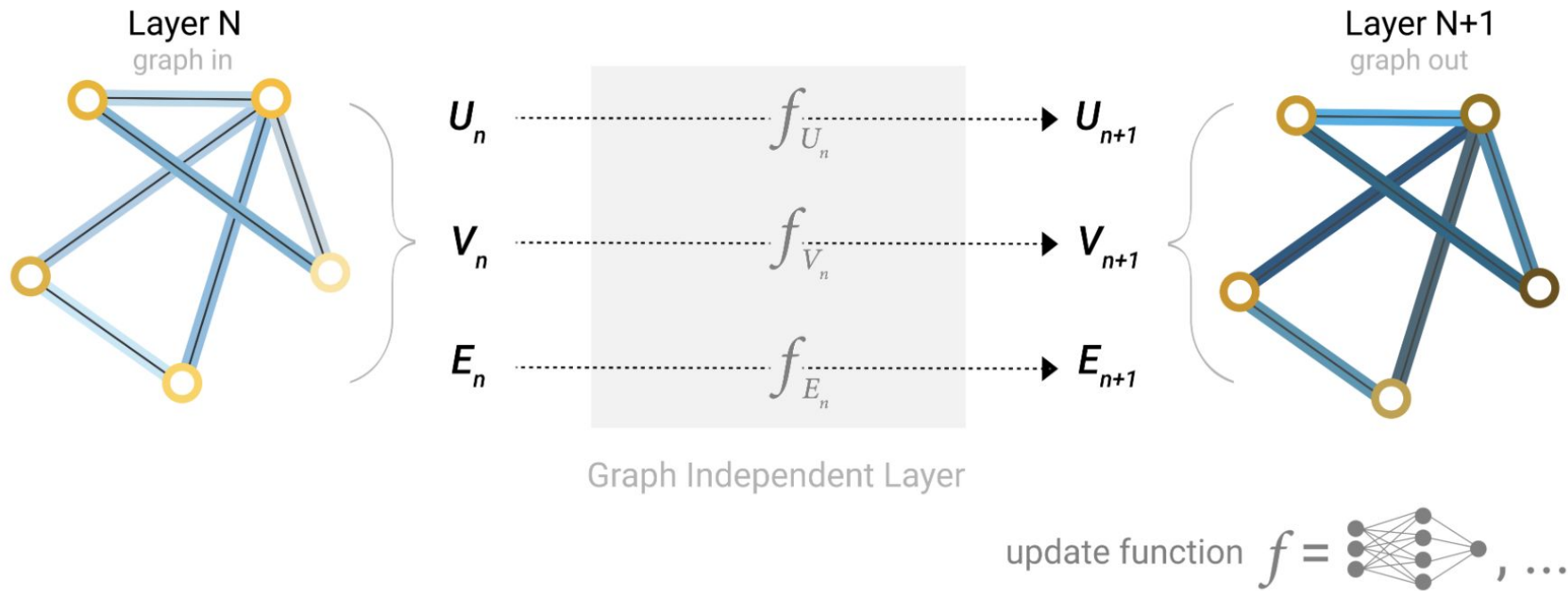
Adjacency List

[[1, 0], [2, 0], [4, 3], [6, 2],  
[7, 3], [7, 4], [7, 5]]

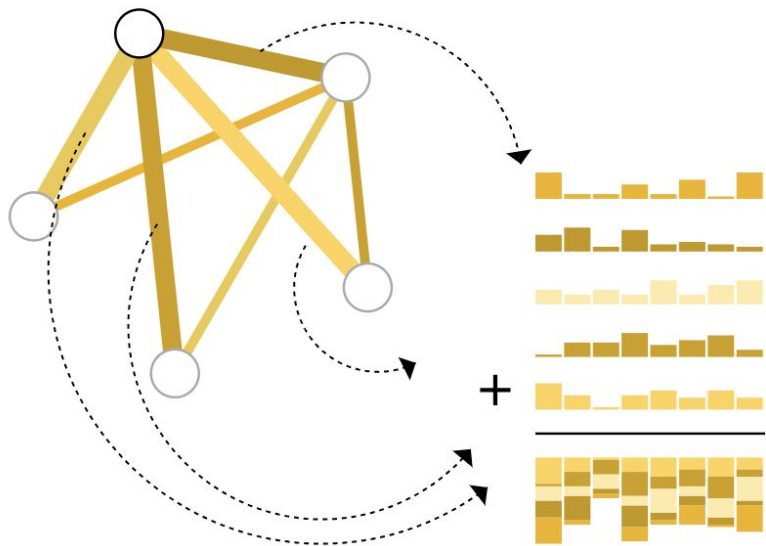
Global

0

# Simple GNN



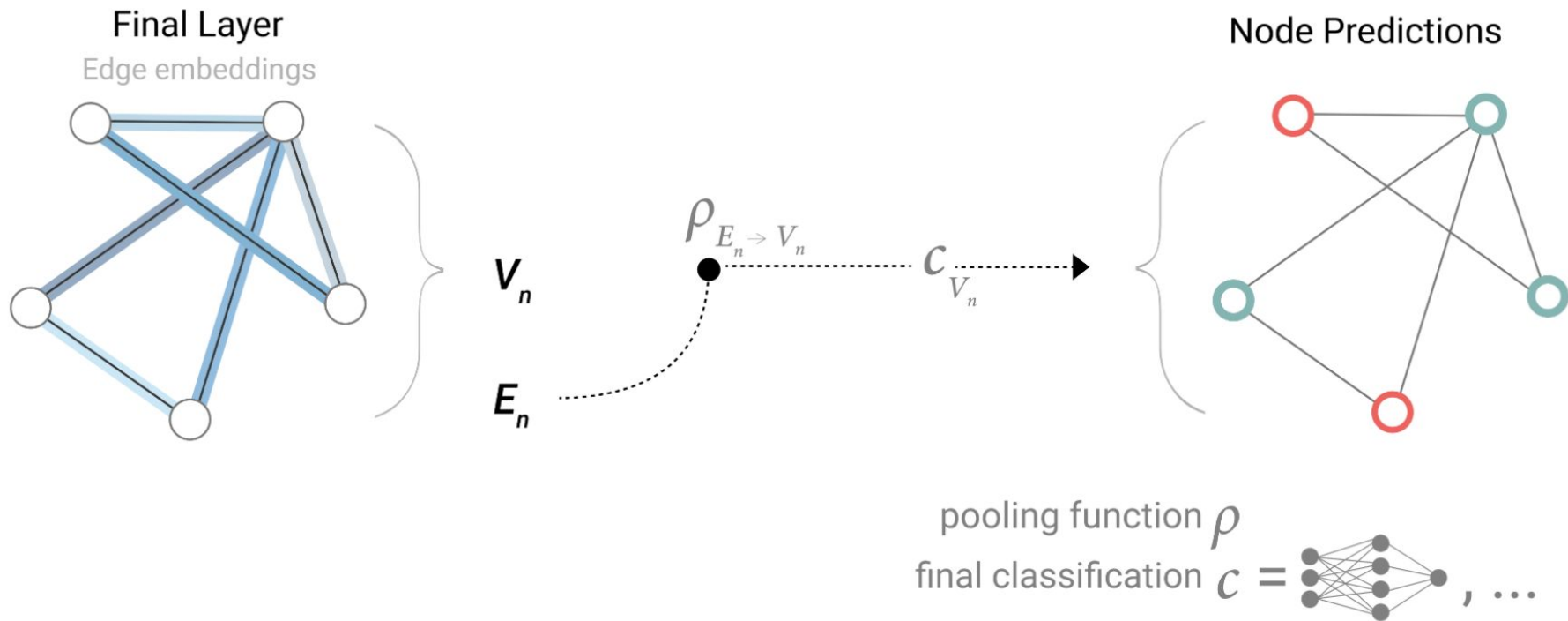
# Aggregation / pooling



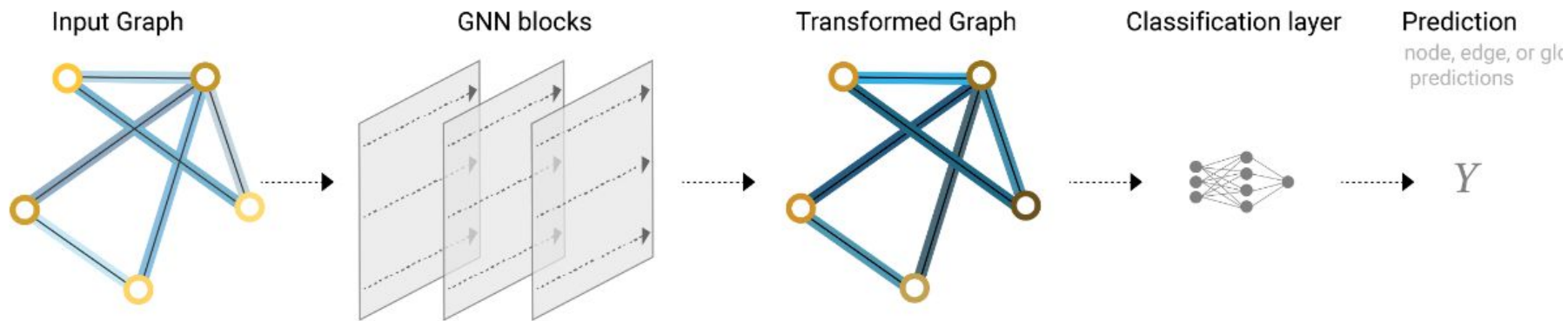
Aggregate information  
from adjacent edges

We represent the *pooling* operation by the letter  $\rho$ , and denote that we are gathering information from edges to nodes as  $\rho_{E_n \rightarrow V_n}$ .

# Aggregation / pooling

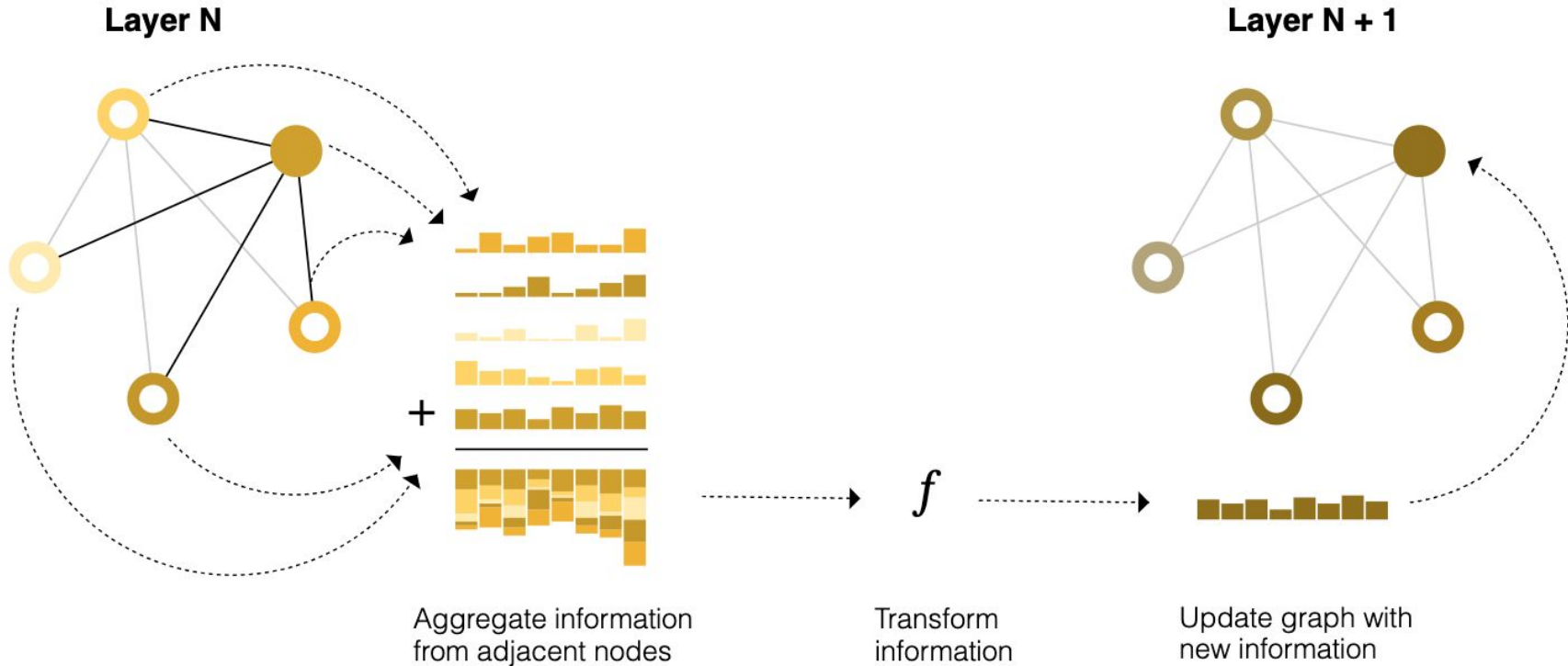


# Classification task

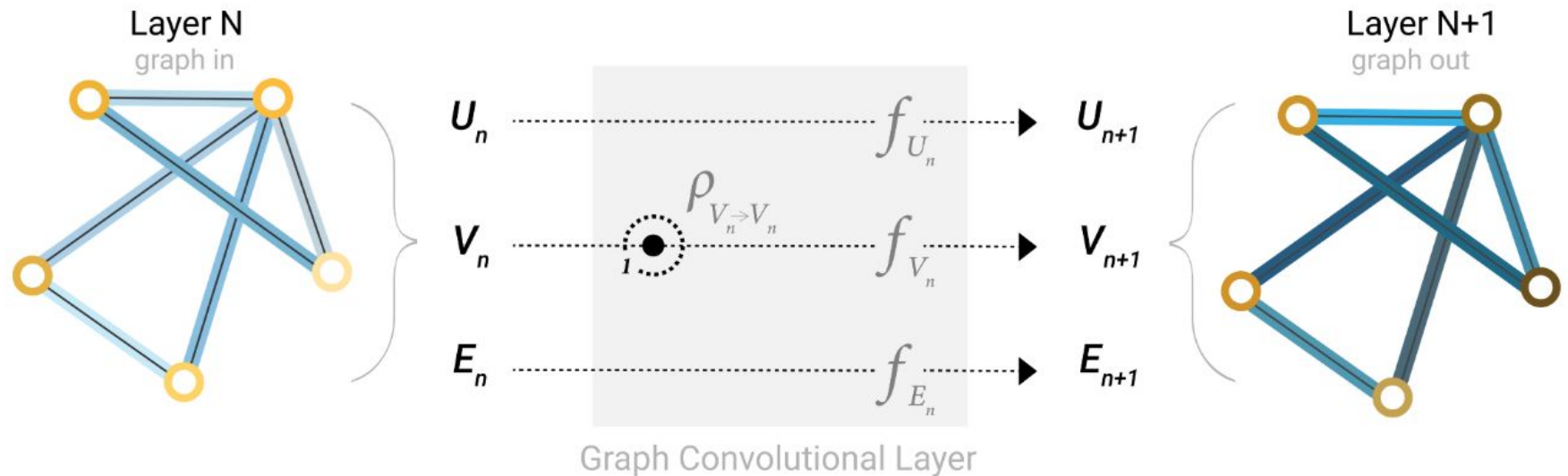


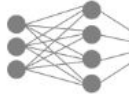
An end-to-end prediction task with a GNN model.

# Aggregation / pooling (in graph layer)

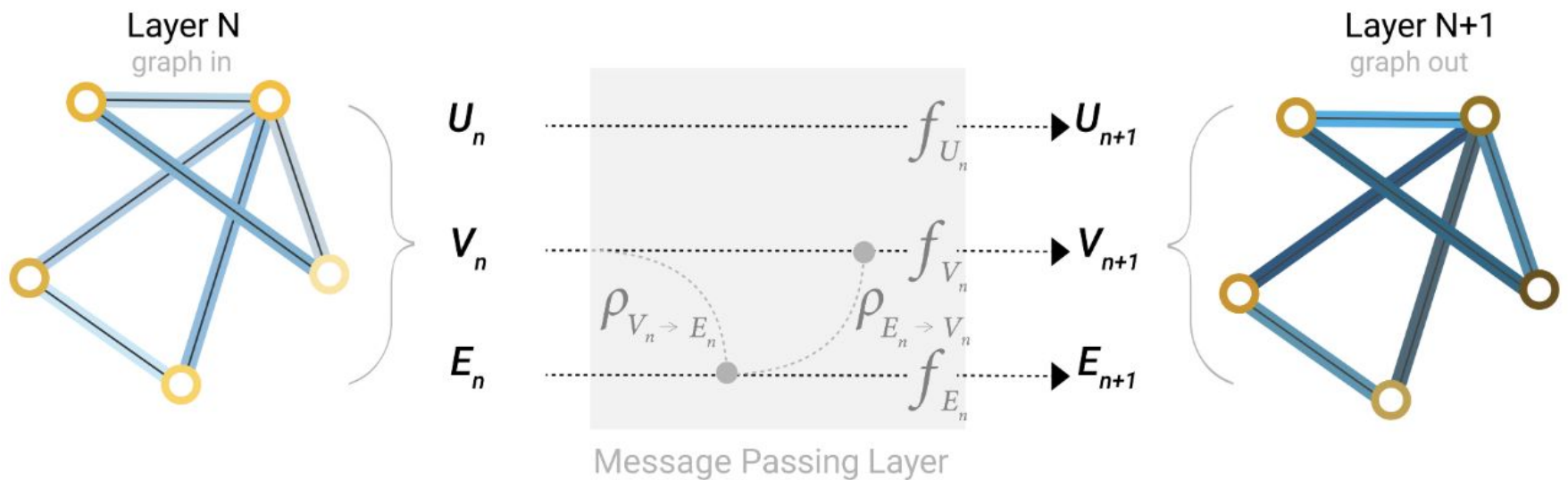


# Aggregation / pooling (in graph layer)



update function  $f =$   , ...  
pooling function  $\rho$

# Aggregation / pooling (in graph layer)



update function  $f =$   , ...

pooling function  $\rho$



---

# Do Transformers Really Perform Bad for Graph Representation?

---

**Chengxuan Ying<sup>1\*</sup>, Tianle Cai<sup>2</sup>, Shengjie Luo<sup>3\*</sup>,  
Shuxin Zheng<sup>4†</sup>, Guolin Ke<sup>4</sup>, Di He<sup>4†</sup>, Yanming Shen<sup>1</sup>, Tie-Yan Liu<sup>4</sup>**

<sup>1</sup>Dalian University of Technology    <sup>2</sup>Princeton University

<sup>3</sup>Peking University    <sup>4</sup>Microsoft Research Asia

yingchengsyuan@gmail.com, tianle.cai@princeton.edu, luosj@stu.pku.edu.cn

{shuz<sup>†</sup>, guoke, dihe<sup>†</sup>, tyliu}@microsoft.com, shen@dlut.edu.cn

# Graph (prelim)

Let  $G = (V, E)$  denote a graph where  $V = \{v_1, v_2, \dots, v_n\}$ ,

neighbors. We denote  $h_i^{(l)}$  as the representation of  $v_i$  at the  $l$ -th layer and define  $h_i^{(0)} = x_i$ . The  $l$ -th iteration of aggregation could be characterized by AGGREGATE-COMBINE step as

$$a_i^{(l)} = \text{AGGREGATE}^{(l)} \left( \left\{ h_j^{(l-1)} : j \in \mathcal{N}(v_i) \right\} \right), \quad h_i^{(l)} = \text{COMBINE}^{(l)} \left( h_i^{(l-1)}, a_i^{(l)} \right), \quad (1)$$

## Entire graph representation

$$h_G = \text{READOUT} \left( \left\{ h_i^{(L)} \mid v_i \in G \right\} \right).$$

## Transformer Prelim

Let  $\hat{H} = [h_1^\top, \dots, h_n^\top]^\top \in \mathbb{R}^{n \times d}$  denote the input of self-attention module

$$Q = H W_Q, \quad K = H W_K, \quad V = H W_V,$$
$$A = \frac{Q K^\top}{\sqrt{d_K}}, \quad \text{Attn}(H) = \text{softmax}(A) V,$$

$$W_Q \in \mathbb{R}^{d \times d_K}, W_K \in \mathbb{R}^{d \times d_K} \text{ and } W_V \in \mathbb{R}^{d \times d_V}.$$

## 3.1 Centrality Encoding

$$A = \frac{QK^\top}{\sqrt{d_K}}, \quad \text{Attn}(H) = \text{softmax}(A) V,$$

- Degree centrality (in and out degree encodings)
- Add embedding vectors (learnable)

$$h_i^{(0)} = x_i + z_{\text{deg}^-(v_i)}^- + z_{\text{deg}^+(v_i)}^+,$$

- If undirected, can unify into one encoding

## 3.2 Spatial Encoding

- Do not have positional information (like seq) in graph
- Encode by creating :  $\phi(v_i, v_j) : V \times V \rightarrow \mathbb{R}$  , to measure spatial info
- choose to be shortest path between two nodes (if connected), else -1

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)},$$

$b_{\phi(v_i, v_j)}$  is a learnable scalar indexed by  $\phi(v_i, v_j)$ , and shared across all layers.

## 3.2 Edge Encoding in Attn

- Take one of shortest path between two nodes  $\mathbf{SP}_{ij} = (e_1, e_2, \dots, e_N)$
- Compute avg of dot products of all edge features + learnable embedding
- Add it as bias term to attention between two vertices

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)} + c_{ij}, \text{ where } c_{ij} = \frac{1}{N} \sum_{n=1}^N x_{e_n} (w_n^E)^T,$$

$x_{e_n}$  is the feature of the  $n$ -th edge  $e_n$  in  $\mathbf{SP}_{ij}$ ,  $w_n^E \in \mathbb{R}^{d_E}$  is the  $n$ -th weight embedding.  
 $d_E$  is the dimensionality of edge feature.

### 3.3 Combining everything

- Apply layer norm (LN) before MHA and MLP

$$h'^{(l)} = \text{MHA}(\text{LN}(h^{(l-1)})) + h^{(l-1)}$$

$$h^{(l)} = \text{FFN}(\text{LN}(h'^{(l)})) + h'^{(l)}$$

-

## 3.3 Master Node

- Add node [VNode] and connect it to all other nodes (like [CLS] token)

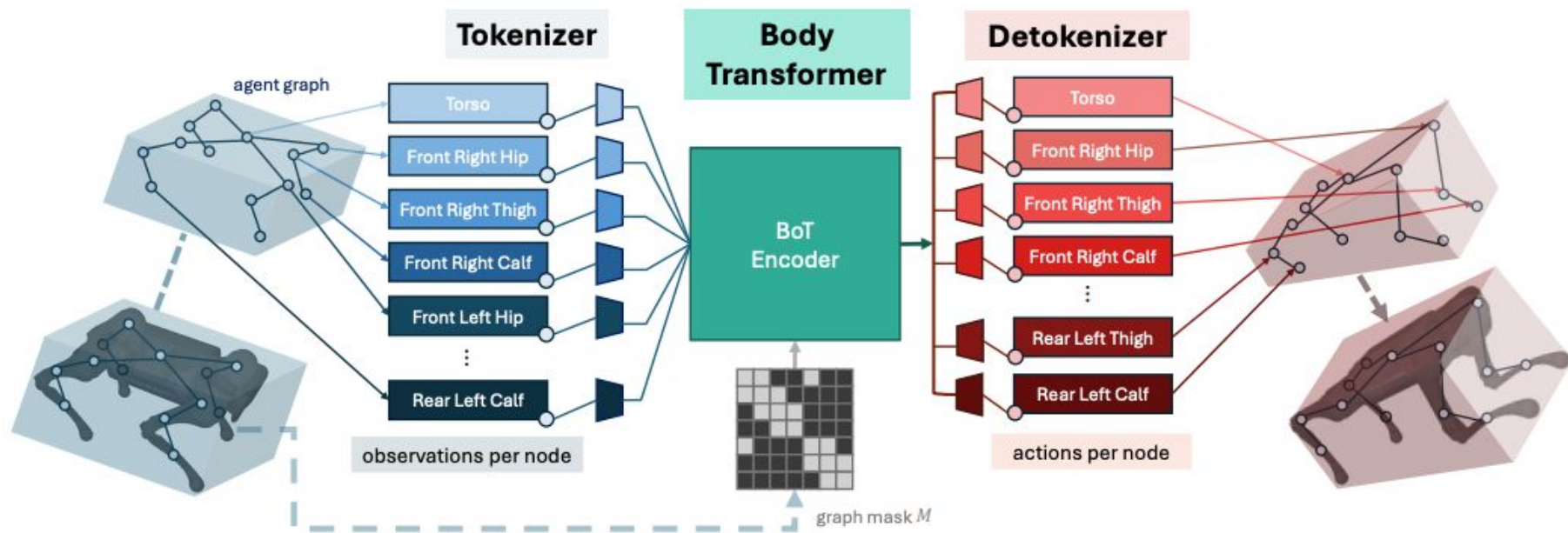
$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)},$$

- Change  $b_{\phi}([VNode], v_j)$  and  $b_{\phi}(v_i, [VNode])$  to be a learnable scalar (not func)



# Body Transformer: Leveraging Robot Embodiment for Policy Learning

Carmelo Sferrazza   Dun-Ming Huang   Fangchen Liu   Jongmin Lee   Pieter Abbeel  
UC Berkeley

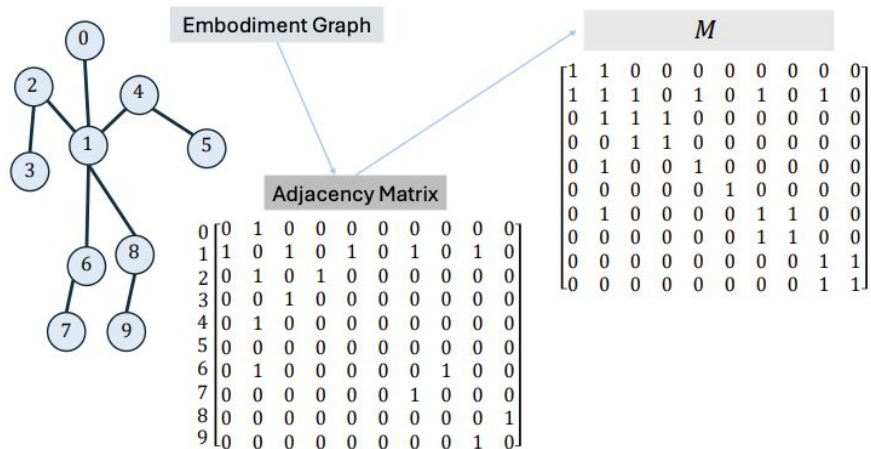


### 3.3 Masked Attention

The attention mechanism can be altered [5] with a binary mask  $M \in \{0, 1\}^{n \times n}$  (where  $n$  is the sequence length), which is equivalent to replacing the elements of  $B$  in (1):

$$B_{i,j} = \begin{cases} 0 & M_{i,j} = 1 \\ -\infty & M_{i,j} = 0 \end{cases},$$

where  $i$  and  $j$  denote row and column indices. This operation effectively results in zeroing out the contribution of the pairs indicated with zeros in the mask  $M$  to the computation of the attention.



### 3.3 Masked Attention

The attention mechanism can be altered [5] with a binary mask  $M \in \{0, 1\}^{n \times n}$  (where  $n$  is the sequence length), which is equivalent to replacing the elements of  $B$  in (1):

$$B_{i,j} = \begin{cases} 0 & M_{i,j} = 1 \\ -\infty & M_{i,j} = 0 \end{cases},$$

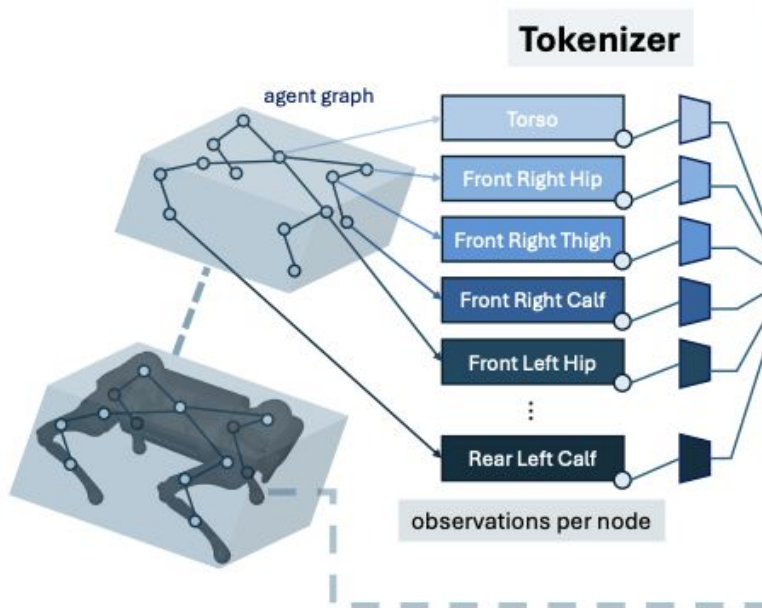
where  $i$  and  $j$  denote row and column indices. This operation effectively results in zeroing out the contribution of the pairs indicated with zeros in the mask  $M$  to the computation of the attention.

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V,$$

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} + B \right) V,$$

## 4.1 BoT Tokenizer

- Map observation  $\rightarrow$  graph local observations
- Assign global quantities to root element, local to nodes represent limbs
- Linear layer projects states at each limb into



## 4.1 BoT Encoder

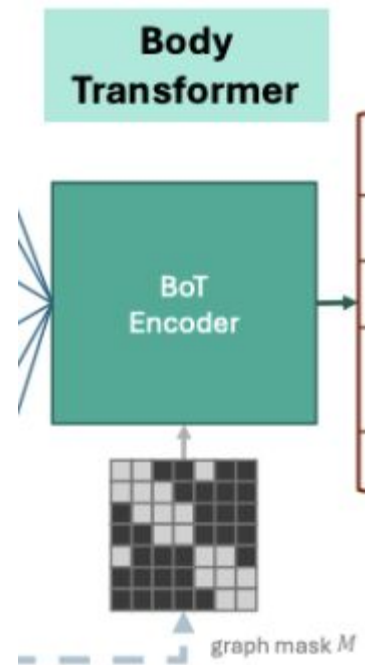
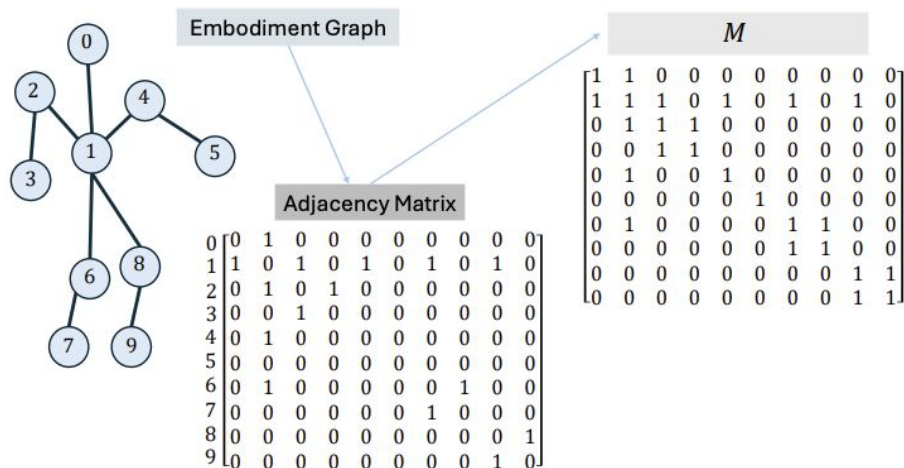
- Alternates between BoT-Hard and BoT-Mix

BoT-Hard:

- Only use matrix  $M$

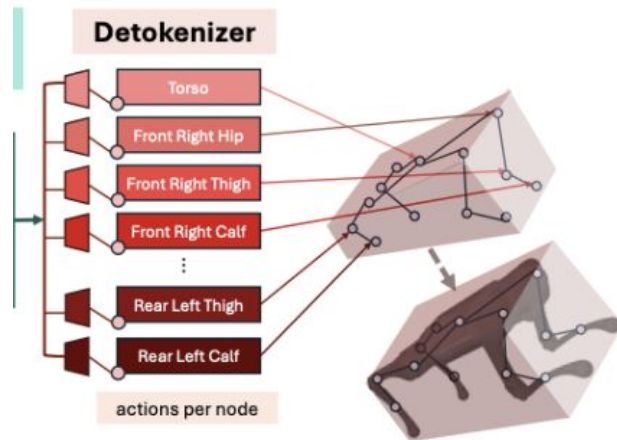
BoT-Mix:

- Alternate between using/ not using matrix  $M$



## 4.1 BoT De-Tokenizer

- Project output from transformer to limbs actions using linear projections
- Linear projections are different for each node



## 5.1 Learning with PPO / RL

maybe?