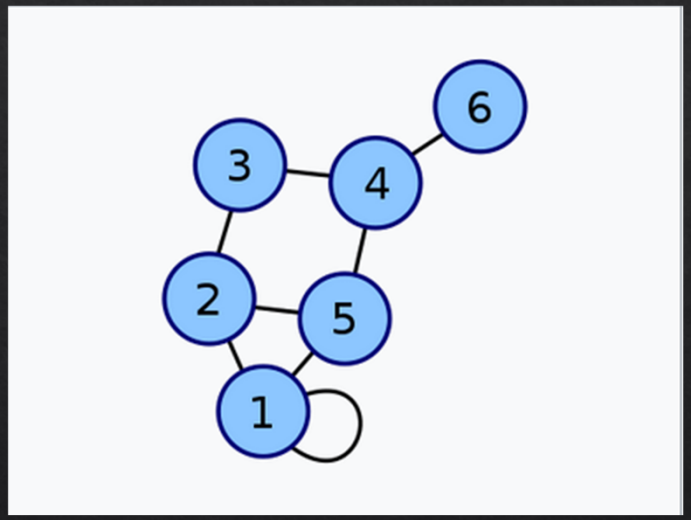
# GNNAdvisor: An Adaptive and Efficient Runtime System for GNN Acceleration on GPUs

# Outline

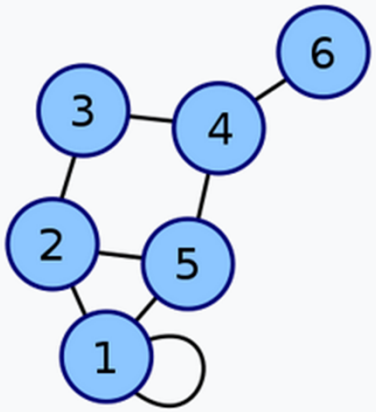
- Background
  - Graphs/Representation
  - Graph Embedding
  - Graph Convolutional Network
  - Graph Isomorphism Network
- GNNAdvisor
  - Loader/Extractor
  - 2D Workload Management
  - Memory Optimization
  - Pipeline
- Performance
  - Inference Performance
  - Training Performance

# Graphs

- Collection of vertices (nodes) and edges (links)
- Represents relations between objects
- Can be directed or undirected
- Applications include protein modeling, social network analysis, linguistics, computer vision



# Graph Representation




**Adjacency matrix**

$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

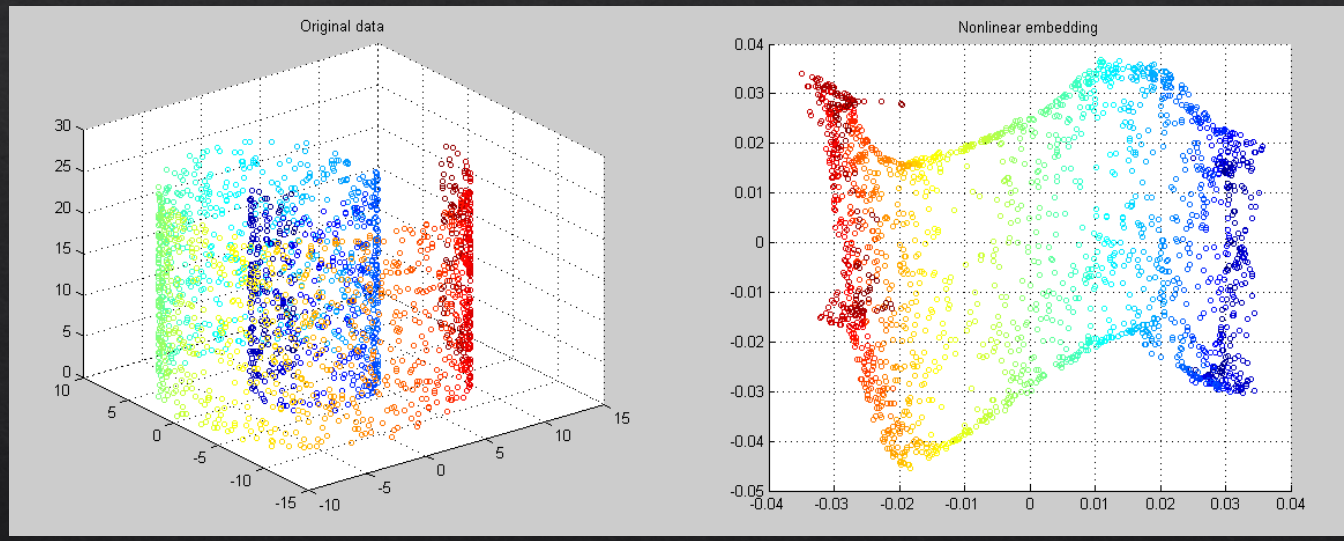
Coordinates are 1–6.

**Degree matrix**

$$\begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

# Graph Embedding

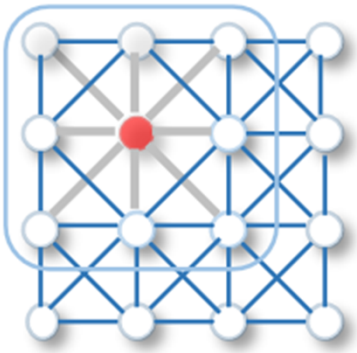
- Reduces the dimension of the graph while maintaining topology
- Allows the use of vector-based tools to compare nodes ($L_P$ norm, cosine similarity, etc.)
- Can be embedded either on graph-level, node-level, or edge-level



# Graph Embedding (How?)

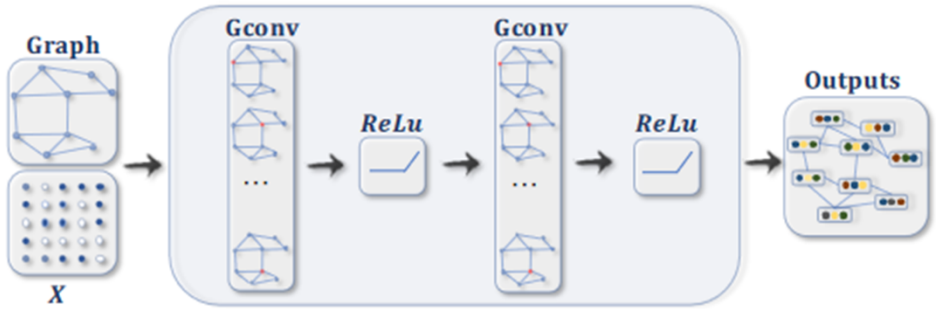
- Traditional Approaches
  - Laplacian Eigenmaps
  - Random Walk
  - PageRank
- Graph Neural Network
  - Graph Convolutional Network (GCN)
  - Graph Isomorphism Network (GIN)

# Graph Convolutional Network



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes the weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of the graph convolutional operation is to take the average value of the node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.



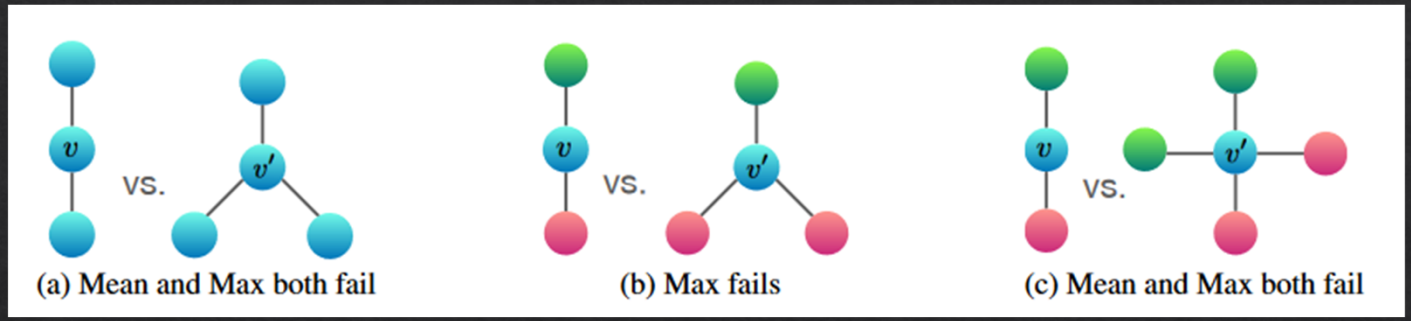

(a) A ConvGNN with multiple graph convolutional layers. A graph convolutional layer encapsulates each node's hidden representation by aggregating feature information from its neighbors. After feature aggregation, a non-linear transformation is applied to the resulted outputs. By stacking multiple layers, the final hidden representation of each node receives messages from a further neighborhood.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

$$h_v^{(k)} = \mathrm{ReLU}\left(W \cdot \mathrm{MEAN}\left\{h_u^{(k-1)},\ \forall u \in \mathcal{N}(v) \cup \{v\}\right\}\right).$$

# Graph Isomorphism Network

- GCN uses the mean of neighbors, but mean is not injective
- Sum neighbors, introduce a weight to the central node, and feed through a fully connected network




(a) Mean and Max both fail (b) Max fails (c) Mean and Max both fail

$$\mathbf{h}_v^{(k)} = MLP((1+\epsilon^{(k)})\mathbf{h}_v^{(k-1)} + \sum_{u \in N(v)} \mathbf{h}_u^{(k-1)}),$$

# GNNAdvisor

- Loader and Extractor – Extract GNN specific information (model type, embedding dimension, graph community)
- Decider – Analytically process the extracted GNN properties to select optimization parameters + reorder nodes
- Kernel&Runtime Crafter – Customize GNNAdvisor's kernel and CUDA runtime based on the Decider's parameter selection

# Loader and Extractor

- GCN/GIN differences
  - GCN prefers memory optimization
  - GIN prefers computing parallelism (aggregation step has greater embedding dimension)
- Node degree/embedding dimension
  - Greater node degree – increase node parallelism
  - Greater embedding dimension – increase parallelism along embedding dimension
- Graph Community
  - Community is when a set of nodes share high intra-group connectivity
  - When embedding size is large, load each community into memory once

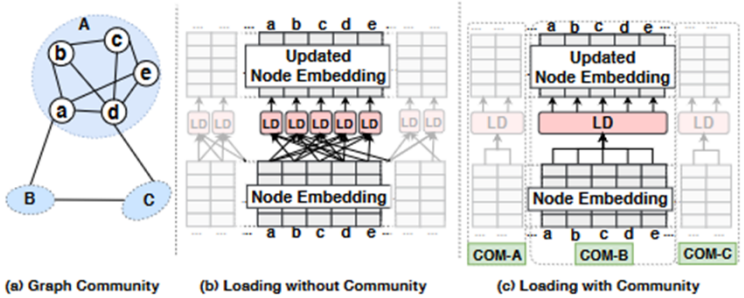
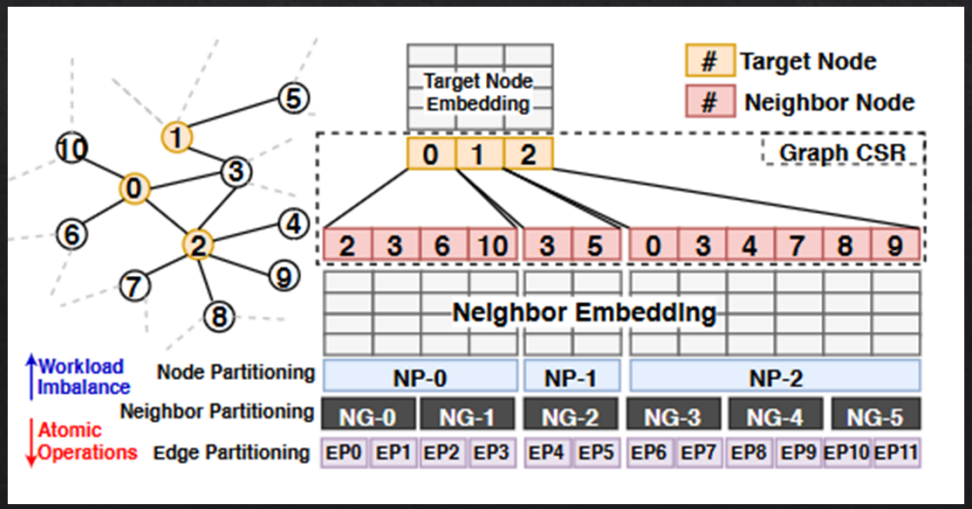



Figure 3: Graph community and its potential benefits. Note that "LD": loading operation. "COM": community.

# 2D Workload Management

- Coarse-grained Neighbor Partitioning
  - Split up node neighbors into fixed-sized groups
  - Mitigate size irregularity for workload units to improve GPU utilization
  - Avoids workload overhead of edge-centric partitioning
  - Introduces a parameter (ngs) to control partition size depending on workload



# 2D Workload Management

- Fine-grained Dimension Partitioning
  - Split up embedding vector calculation across several workers
  - Allows for larger sizes of embedding dimension to be efficiently computed
  - Introduces a parameter (dw) that sets how many workers are partitioned across an embedding

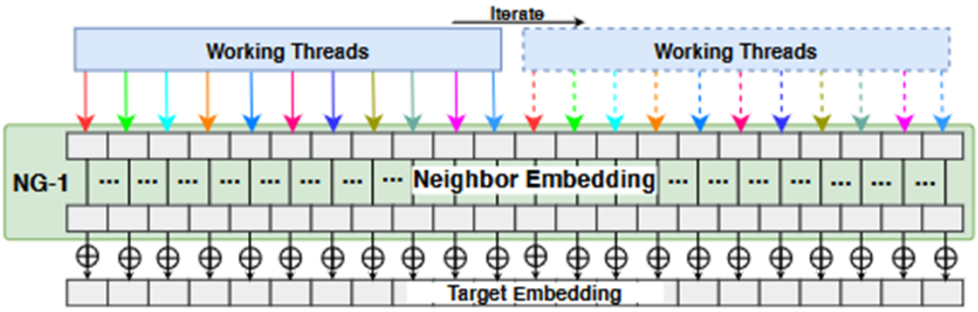
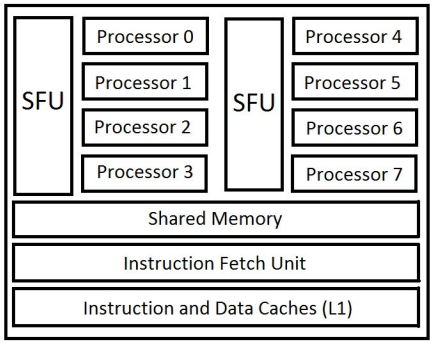



Figure 5: Dimension Partitioning. $\oplus$: Accumulated add.

# 2D Workload Management

- Warp-based Thread Alignment
  - A Warp is a set of 32 threads running on a streaming multiprocessor. All threads execute the same instruction
  - Align neighbor-group partitions to each warp
  - Inter-thread synchronization is minimized since each neighbor-group thread is working on a different section of embedding vector
  - Balanced workload across warps allows for improved SM scheduler performance
  - Warp threads access contiguous memory locations therefore merging global memory accesses



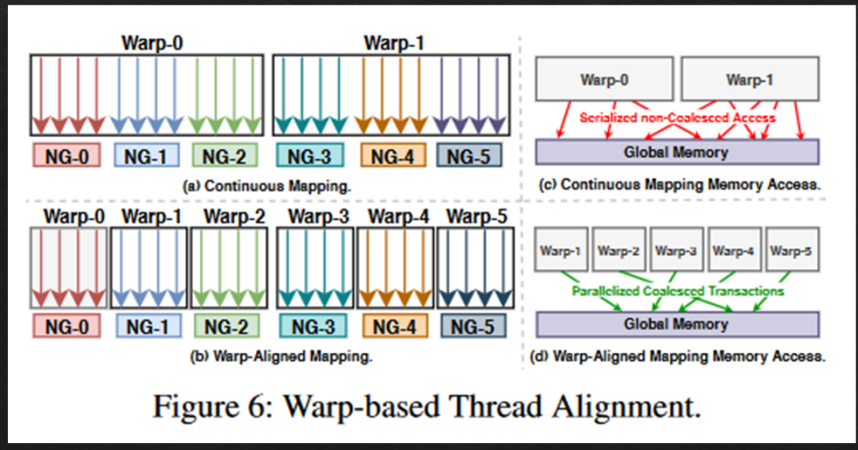
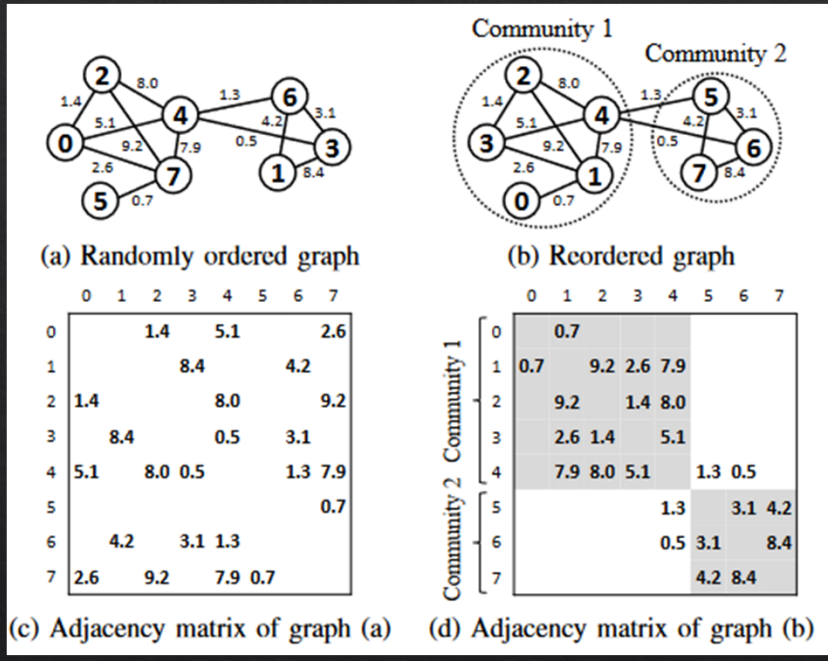




Figure 6: Warp-based Thread Alignment.

# Memory Optimization

- Community-aware Node Renumbering
  - Assign community node IDs as close as possible
  - Nodes with close IDs are scheduled close to each other
  - Increased chance of community nodes scheduled on one SM meaning shared L1/L2 cache
  - Uses Rabbit Order to conduct reorder
  - If $\sqrt{AES} > \lfloor \frac{\sqrt{\#N}}{100} \rfloor$ then perform reorder

$$\mathbf{AES} = \frac{1}{\#E} \sum_{(src_{id}, trg_{id}) \in E} |src_{id} - trg_{id}|$$



# Memory Optimization

- Warp-aware Memory Customization
  - If each warp places its result into global memory, then for each target node with k neighbor groups and ngs nodes per group, there are O(k*ngs*Dim) memory accesses.
  - Instead, take advantage of shared L1 and L2 memory between communities
  - Each thread block (collection of warps running on a SM) designates one warp leader that copies over the intermediate result to global memory
  - A reserved section of shared memory is set aside for each warp to place their result in
  - This reduces global memory access to O(Dim)

# Decider

$$\mathbf{WPT} = ngs \times \frac{Dim}{dw}, \quad \mathbf{SMEM} = \frac{tpb}{tpw} \times Dim \times FloatS$$

$$dw = \begin{cases} tpw & Dim \geq tpw \\ \frac{tpw}{2} & Dim < tpw \end{cases}$$

ngs: neighbor-group size

Dim: size of embedding dimension

dw: workers per dimension

tpb: threads per block

tpw: threads per warp (32)

FloatS: Float size (4)

Workers per Thread (WPT) should be made to be about 1024.

SMEM < SMEMperBlock = 48kB – 96kB

tpb is a power of 2 but less than 1024

# Final Pipeline

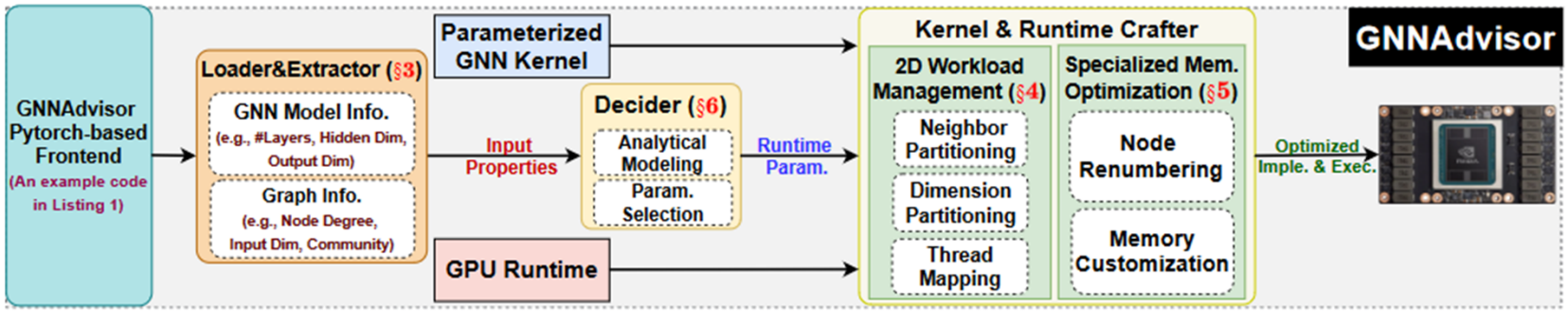



Figure 1: Overview of GNNAdvisor.

# Performance

- Three types of graph
  - Type 1:  Small in nodes and edges, but large embedding dimension (Tall)
  - Type 2: Set of small, disjoint graphs
  - Type 3: Large in nodes and edges, small in embedding dimension (Wide)
- Compared to DGL

Table 1: Datasets for Evaluation.

| Type | Dataset | #Vertex | #Edge | Dim. | #Class |
|---|---|---|---|---|---|
| I | Citeseer | 3,327 | 9,464 | 3,703 | 6 |
| | Cora | 2,708 | 10,858 | 1,433 | 7 |
| | Pubmed | 19,717 | 88,676 | 500 | 3 |
| | PPI | 56,944 | 818,716 | 50 | 121 |
| II | PROTEINS_full | 43,471 | 162,088 | 29 | 2 |
| | OVCAR-8H | 1,890,931 | 3,946,402 | 66 | 2 |
| | Yeast | 1,714,644 | 3,636,546 | 74 | 2 |
| | DD | 334,925 | 1,686,092 | 89 | 2 |
| | TWITTER-Partial | 580,768 | 1,435,116 | 1,323 | 2 |
| | SW-620H | 1,889,971 | 3,944,206 | 66 | 2 |
| III | amazon0505 | 410,236 | 4,878,875 | 96 | 22 |
| | artist | 50,515 | 1,638,396 | 100 | 12 |
| | com-amazon | 334,863 | 1,851,744 | 96 | 22 |
| | soc-BlogCatalog | 88,784 | 2,093,195 | 128 | 39 |
| | amazon0601 | 403,394 | 3,387,388 | 96 | 22 |

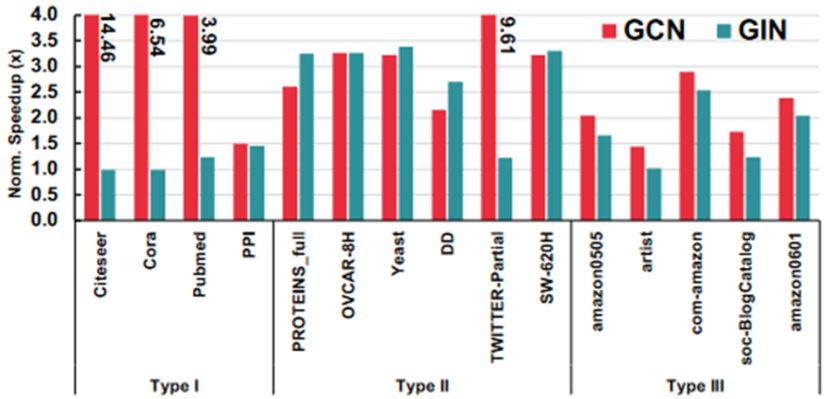



Figure 8: Inference speedup (×) over DGL on GCN and GIN.

# Training Speedup

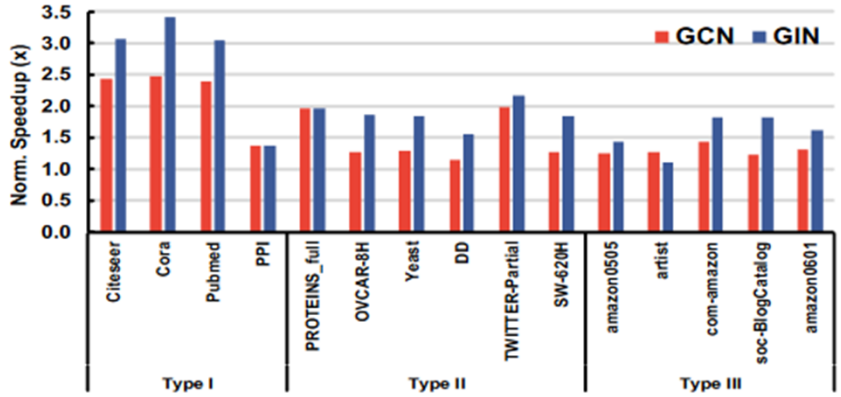



Figure 9: Training speedup (×) over DGL on GCN and GIN.

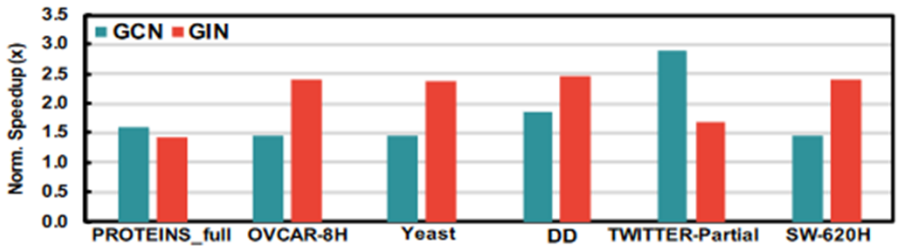



Figure 10: Training speedup (×) over PyG on GCN and GIN.

# Questions?