# Scaling Up Graph Neural Network

PRESENTER: **Liao Ningyi**

SUPERVISOR: **Asst Prof Luo Siqiang**

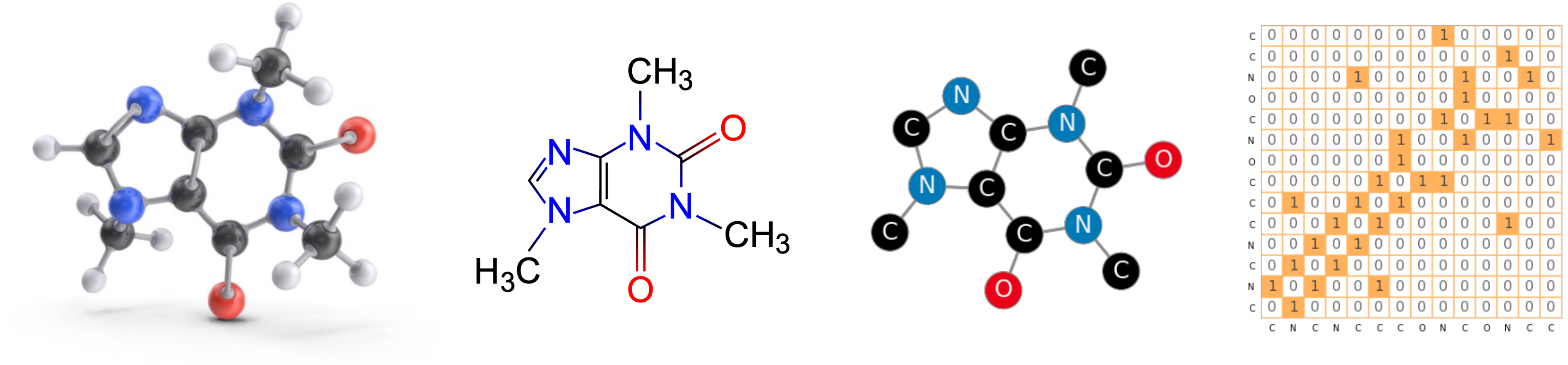
27 Jun 2023

# Contents

# Contents

# Graph: A Ubiquitous Data Structure

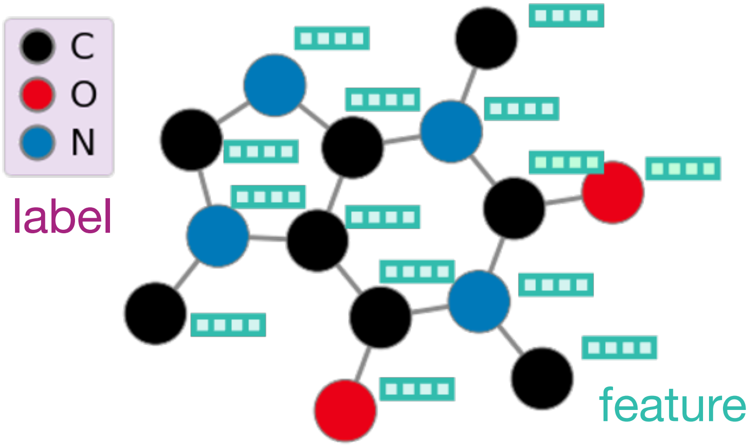
- Graphs model entities (**nodes**) and relationship (**edges**)
- Graph data structures are **non-Euclidean**



Different representations of the caffeine molecule

# Graph Representation

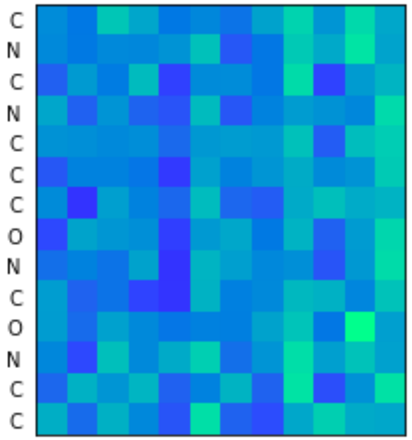
- Nodes $V$, Edges $E$
- Each node: Feature $\boldsymbol{x}$ , Label $y$
- Adjacency matrix $\boldsymbol{A}$ , Feature matrix $\boldsymbol{X}$




Attributed Graph

| | C | N | C | N | C | C | C | O | N | C | O | N | C | C |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| N | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| O | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| N | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| O | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| N | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

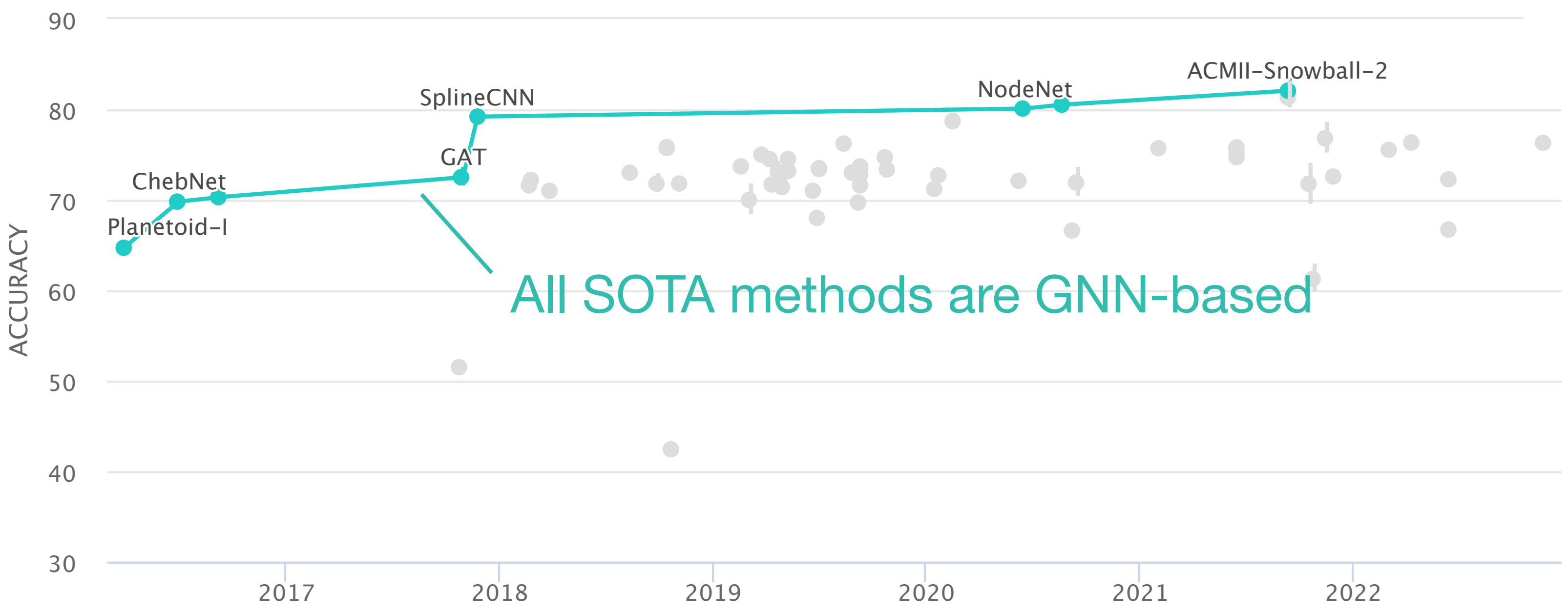
Adjacency matrix




Feature matrix

# GNN: Graph Neural Network

- Apply deep learning architectures to graph data
- Achieve strong performance on graph learning tasks

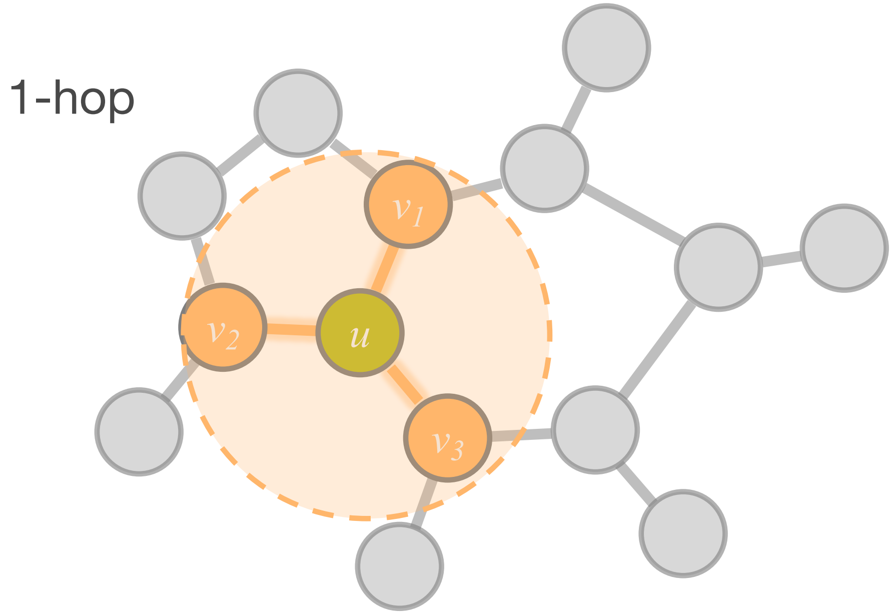


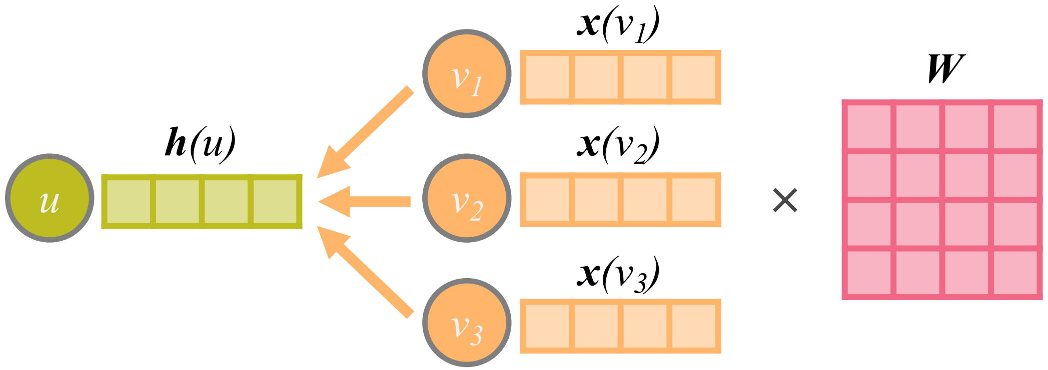

Leaderboard of Node Classification on Citeseer

[1] https://paperswithcode.com/sota/ (2023).

# GNN: Graph Neural Network

- Graph convolution: aggregate neighbor information $N(u)$ by learnable weights $\boldsymbol{W}$ to each node as node representation $\boldsymbol{h}$



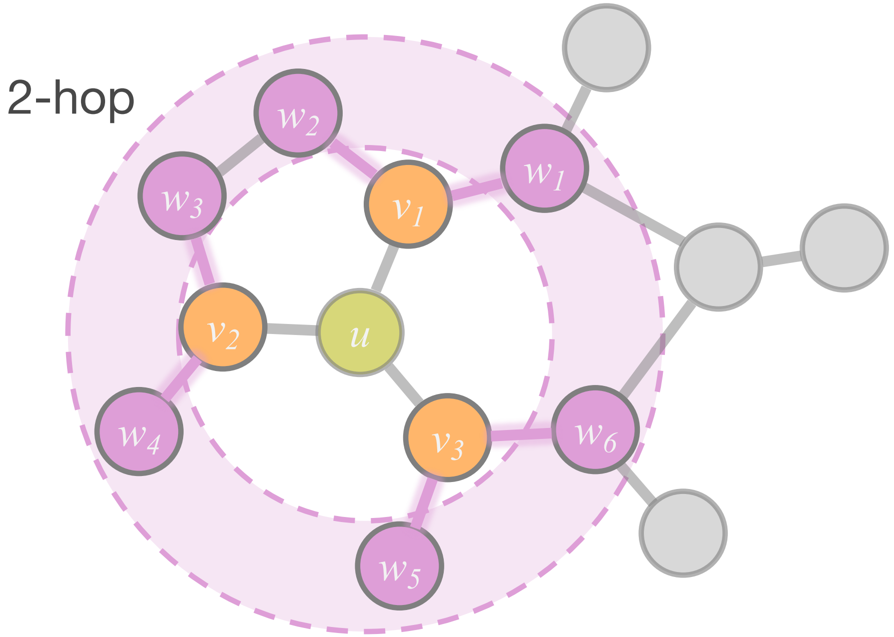



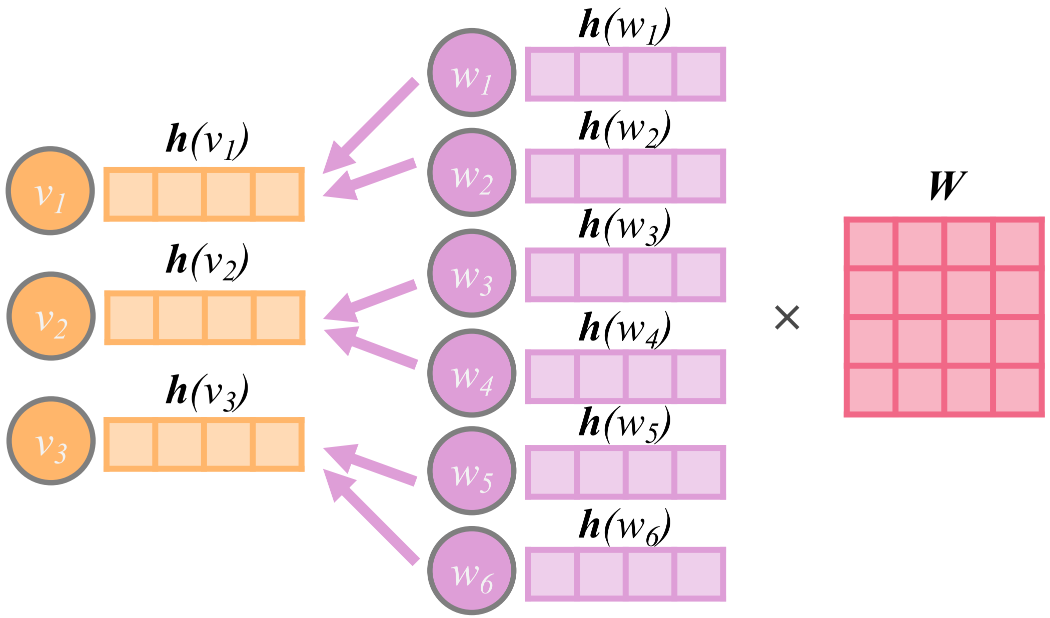

$$\boldsymbol{h}(u) = \sum_{v \in N(u)} \boldsymbol{x}(v) \cdot \boldsymbol{W}$$

[1] T Kipf & M Welling. "Semi-supervised classification with graph convolutional networks". ICLR 2017.

# GNN: Graph Neural Network

- Stacking multi-hop graph convolutions as <u>layers</u>



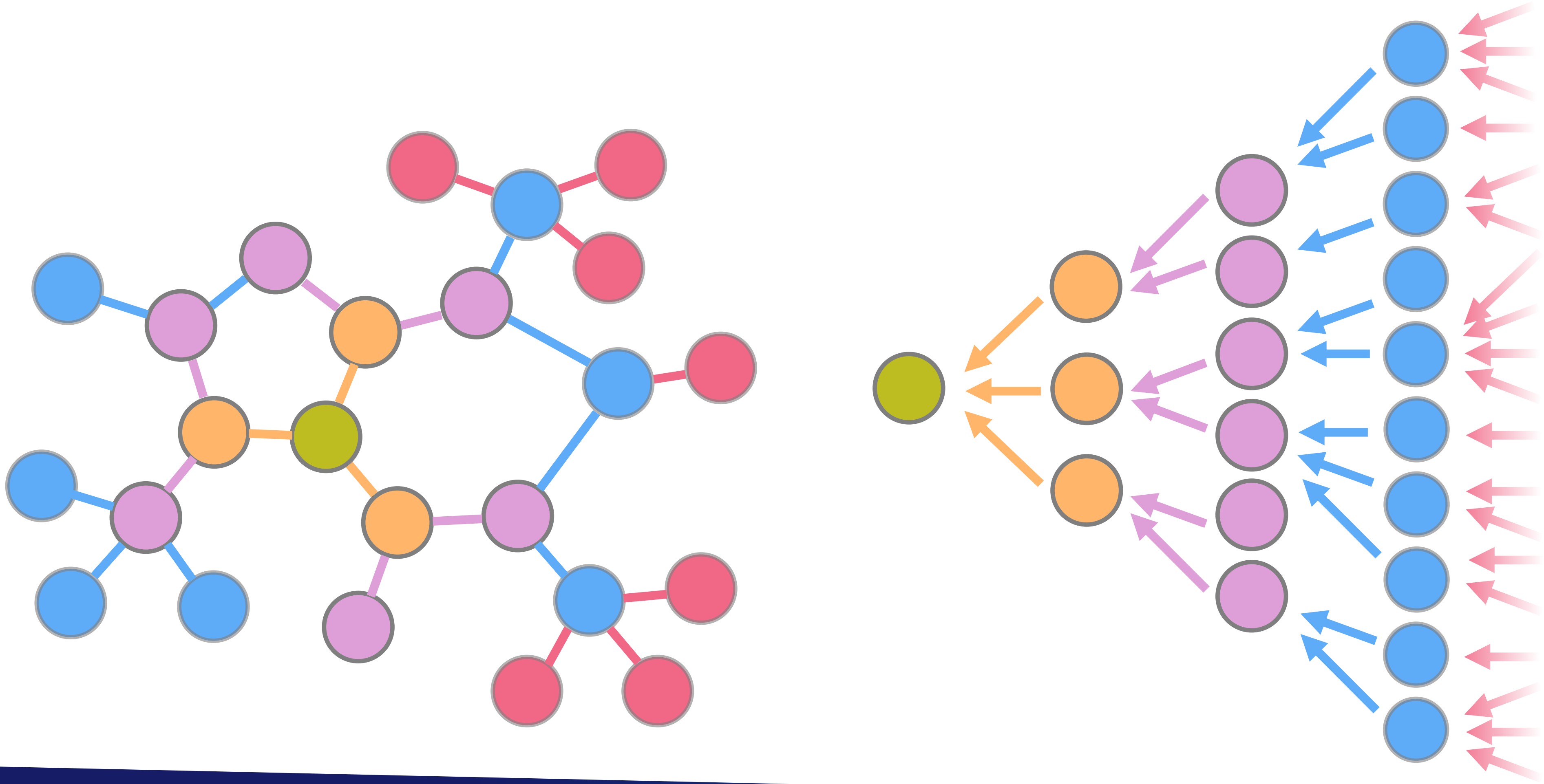




$$\boldsymbol{h}(v) = \sum_{w \in N(v)} \boldsymbol{h}(w) \cdot \boldsymbol{W}$$

[1] T Kipf & M Welling. “Semi-supervised classification with graph convolutional networks”. ICLR 2017.

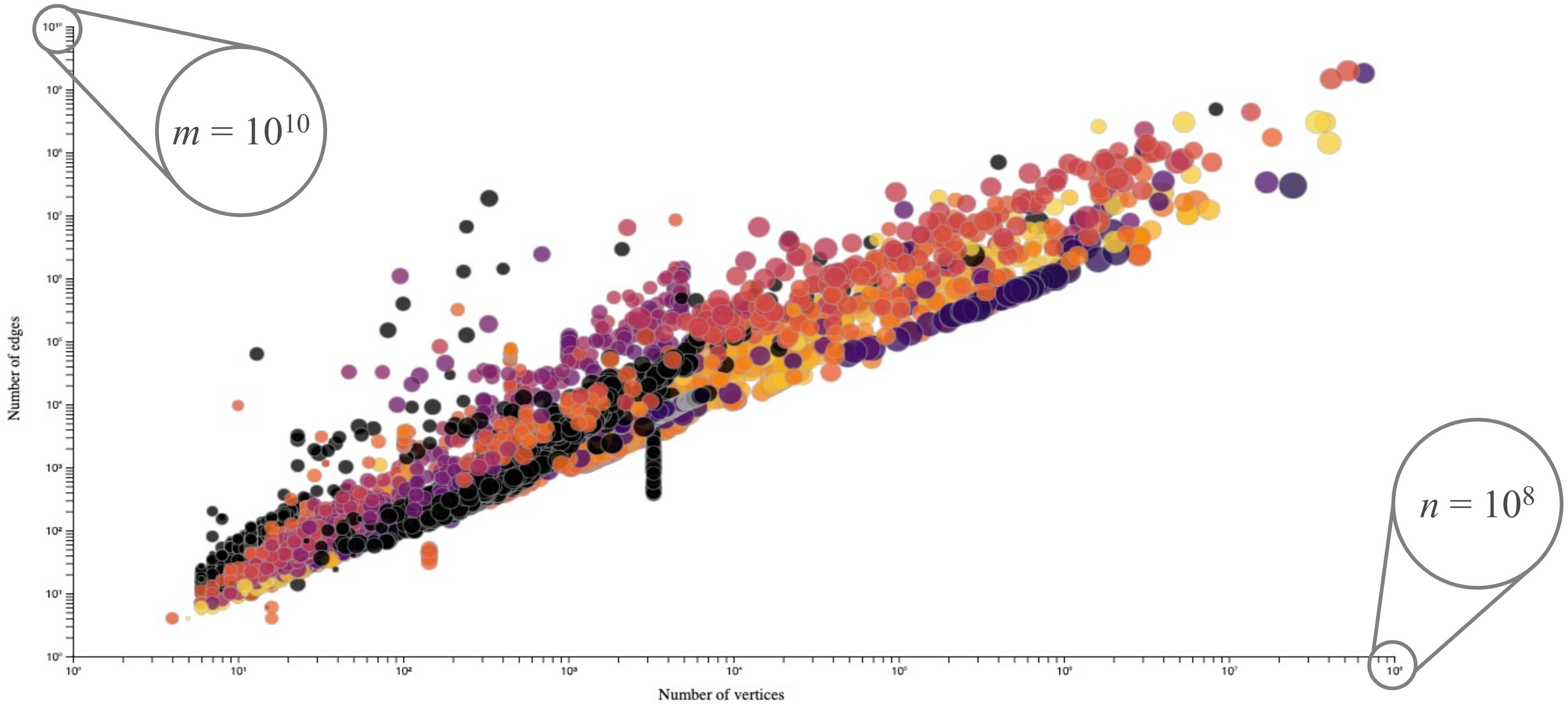
# Challenge: GNN Scalability

- Curse of scale: computation overhead greatly increase!



# Challenge: GNN Scalability

- Modern real-world graphs are on the scale of millions or billions




Scales of recent graph datasets

[1] T Peixoto, "The Netzschleuder network catalogue and repository", https://networks.skewed.de/ (2020).

# Motivations and Objectives

- How to scale up GNN to learn million- or billion-scale graphs?
  - What are the **bottlenecks of GNN scalability** with respect to computation time and memory complexity?
  - How to simplify and optimize **GNN graph propagation** using advanced graph management techniques?
  - How to address the issue of GNN scalability in different **variants of graph data**?

# SCARA: Scalable GNN with Feature-Oriented Optimization

**Ningyi Liao***, Dingheng Mo*, Siqiang Luo, Xiang Li, Pengcheng Yin. "SCARA: Scalable Graph Neural Networks with Feature-Oriented Optimization". *Proceedings of the VLDB Endowment*, Vol. 15, No. 11, pp. 3240-3248, 2022.

# Contents

# Scalability Issue of GNNs

- GNN in matrix form:

Feature Matrix

$$\boldsymbol{H}^{(0)} = \boldsymbol{X}$$

$$\boldsymbol{H}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right), \quad l = 0, 1, \cdots, L-1$$

Adjacency Matrix    Layer Representation    Learnable Weights

Graph Propagation    Feature Transformation

[1] T Kipf & M Welling. “Semi-supervised classification with graph convolutional networks”. ICLR 2017.

# Scalability Issue of GNNs

- Bottleneck of GNN computation is graph propagation

$$\boldsymbol{H}^{(l+1)} = \sigma\left(\tilde{\boldsymbol{A}}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}\right), \quad l = 0, 1, \cdots, L-1$$

Adjacency Matrix $n \times n$

Layer Representation $n \times F$

Learnable Weights $F \times F$

$$\underbrace{n \times n \qquad n \times F}_{\text{Graph Propagation}} \quad \underbrace{\qquad F \times F}_{\text{Feature Transformation}}$$

Time Complexity: Graph Propagation $O(LmF)$; Feature Transformation $O(LnF^2)$

Graph Propagation:
- Sparse-Dense Matrix Mul ☹ *Not optimized for minibatch*
- $m$ at a larger scale than $n$ ☹ *Not scalable to large $m$ and $n$*

Feature Transformation:
- Dense-Dense Matrix Mul
- Similar to common NN

[1] T Kipf & M Welling. “Semi-supervised classification with graph convolutional networks”. ICLR 2017.
[2] W Chiang et al. “Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks”. KDD 2019.

# Existing Work: Decoupling

- GCN: iterative propagation and transformation

$$\boldsymbol{H}^{(L)} = \sigma(\underbrace{\tilde{\boldsymbol{A}}\sigma(\tilde{\boldsymbol{A}}\cdots\tilde{\boldsymbol{A}}\boldsymbol{X}}_{L\text{ Graph Propagation}}\underbrace{\boldsymbol{W}^{(1)}\cdots)\boldsymbol{W}^{(L-1)})}_{L\text{ Feature Transformation}}$$

- SGC: decoupled propagation as precomputation

$$\underbrace{\boldsymbol{H}^{(0)} = \boldsymbol{P}}_{\text{Graph Embedding}} = \underbrace{\tilde{\boldsymbol{A}}^{L}\cdot\boldsymbol{X}}_{\text{Precomputed Propagation}}$$

$$\boldsymbol{H}^{(l+1)} = \sigma(\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \quad l = 0, 1, \cdots, L-1$$

[1] T Kipf & M Welling. “Semi-supervised classification with graph convolutional networks”. ICLR 2017.
[2] F Wu et al. “Simplifying graph convolutional networks”. ICML 2019.

# Existing Work: Decoupling

- Time complexity of SGC is still not scalable

$$\boldsymbol{P} = \tilde{\boldsymbol{A}}^L \cdot \boldsymbol{X} = \tilde{\boldsymbol{A}}\Big(\tilde{\boldsymbol{A}} \cdots \big(\underbrace{\tilde{\boldsymbol{A}}}_{\substack{\text{Adjacency Matrix} \\ n \times n}}\underbrace{\boldsymbol{X}}_{\substack{\text{Feature Matrix} \\ n \times F}}\big)\Big)$$

- $L$ Sparse-Dense Matrix Mul → Time Complexity: $O(LmF)$
- Result stored in $P$ → Memory Complexity: $O(nF)$

[1] F Wu et al. “Simplifying graph convolutional networks”. ICML 2019.

# Our Model: SCARA

- Propagation as precomputation

$$\boldsymbol{H}^{(0)} = \boldsymbol{P} = \sum_{l=0}^{\infty} \alpha(1-\alpha)^l \tilde{\boldsymbol{A}}^l \cdot \boldsymbol{X}$$

Time Complexity: $O(F\sqrt{m \log n}/\lambda)$

Memory Complexity: $O(nF)$

☺ *Only sublinear to $m$*

☺ *Efficient for CPU precomputation*

- Feature transformation

$$\boldsymbol{H}^{(l+1)} = \sigma(\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)}), \quad l = 0, 1, \cdots, L-1$$

Time Complexity: $O(LnF^2)$

Memory Complexity: $O(Ln_bF + LF^2)$

☺ *Efficient GPU training*

# Our Model: SCARA

- **Time Complexity**:
  - Sub-linear precomputation time complexity
  - Efficient decoupled training and inference

TABLE 2.1: Precomputation, training, and inference time complexity of common GNN models.

| Model | Precomp. Time | Training Time | Inference Time |
|---|---|---|---|
| GCN [3] | – | $O(ILmF + ILnF^2)$ | $O(LmF + LnF^2)$ |
| Cluster-GCN [22] | $O(m)$ | $O(ILmF + ILnF^2)$ | $O(LmF + LnF^2)$ |
| GraphSAINT [23] | – | $O(IL_PLnF^2)$ | $O(LmF + LnF^2)$ |
| GAS [24] | $O(m + LnF)$ | $O(ILmF + ILnF^2)$ | $O(nF)$ |
| APPNP [25] | $O(m)$ | $O(IL_PmF + ILnF^2)$ | $O(L_PmF + LnF^2)$ |
| PPRGo [26] | $O(m/\delta)$ | $O(IKnF + ILnF^2)$ | $O(KnF + LnF^2)$ |
| SGC [27] | $O(L_PmF)$ | $O(ILnF^2)$ | $O(LnF^2)$ |
| GBP [11] | $O(L_PF\sqrt{L_Pm\log(L_Pn)}/\epsilon)$ | $O(ILnF^2)$ | $O(LnF^2)$ |
| **SCARA (ours)** | $O(F\sqrt{m\log n}/\lambda)$ | $O(ILnF^2)$ | $O(LnF^2)$ |

*Not scalable to large $m$ and $n$*

# Our Model: SCARA

- **Memory Complexity**:
  - Efficient precomputation memory usage
  - Minibatch training and inference

TABLE 2.2: Precomputation, training, and inference memory complexity of common GNN models.

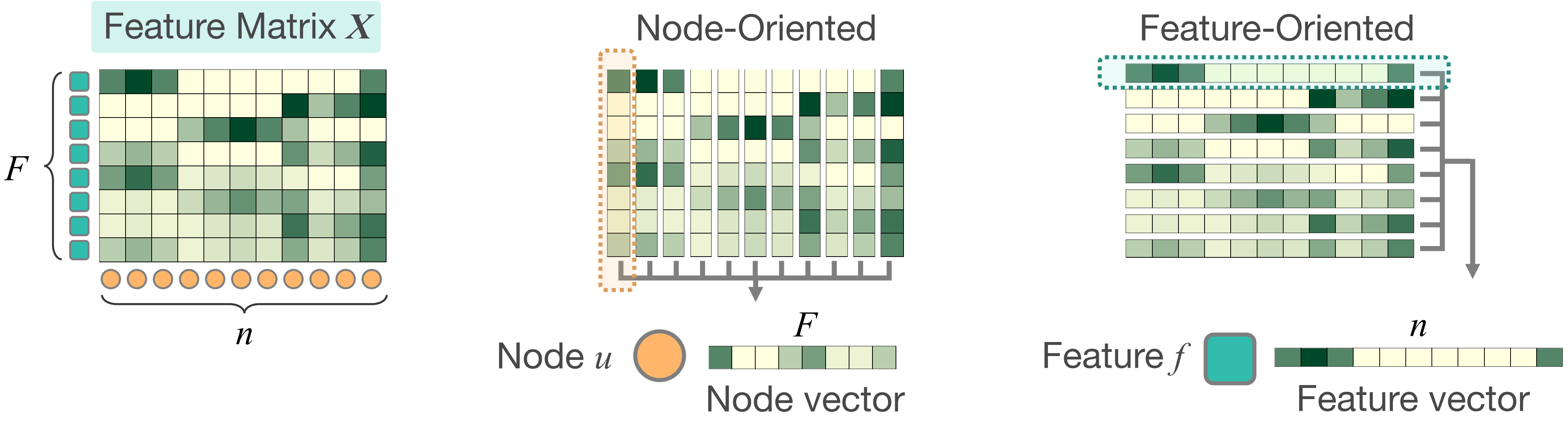
| Model | Precomp. Mem. | Training Mem. | Inference Mem. |
|---|---|---|---|
| GCN [3] | – | $O(LnF + LF^2)$ | $O(LnF + LF^2)$ |
| Cluster-GCN [22] | $O(n)$ | $O(Ln_bF + LF^2)$ | $O(LnF + LF^2)$ |
| GraphSAINT [23] | – | $O(L_PLn_bF + LF^2)$ | $O(LnF + LF^2)$ |
| GAS [24] | $O(LnF)$ | $O(Ldn_bF + LF^2)$ | $O(Ldn_bF + LF^2)$ |
| APPNP [25] | $O(m)$ | $O(Ln_bF + LF^2 + nn_b)$ | $O(Ln_bF + LF^2 + nn_b)$ |
| PPRGo [26] | $O(n/\delta)$ | $O(Ln_bF + LF^2 + Kn_b)$ | $O(Ln_bF + LF^2 + Kn_b)$ |
| SGC [27] | $O(m)$ | $O(Ln_bF + LF^2)$ | $O(Ln_bF + LF^2)$ |
| GBP [11] | $O(nF)$ | $O(Ln_bF + LF^2)$ | $O(Ln_bF + LF^2)$ |
| **SCARA (ours)** | $O(nF)$ | $O(Ln_bF + LF^2)$ | $O(Ln_bF + LF^2)$ |

*Not suitable for minibatch*

*Not scalable to large $m$ and $n$*

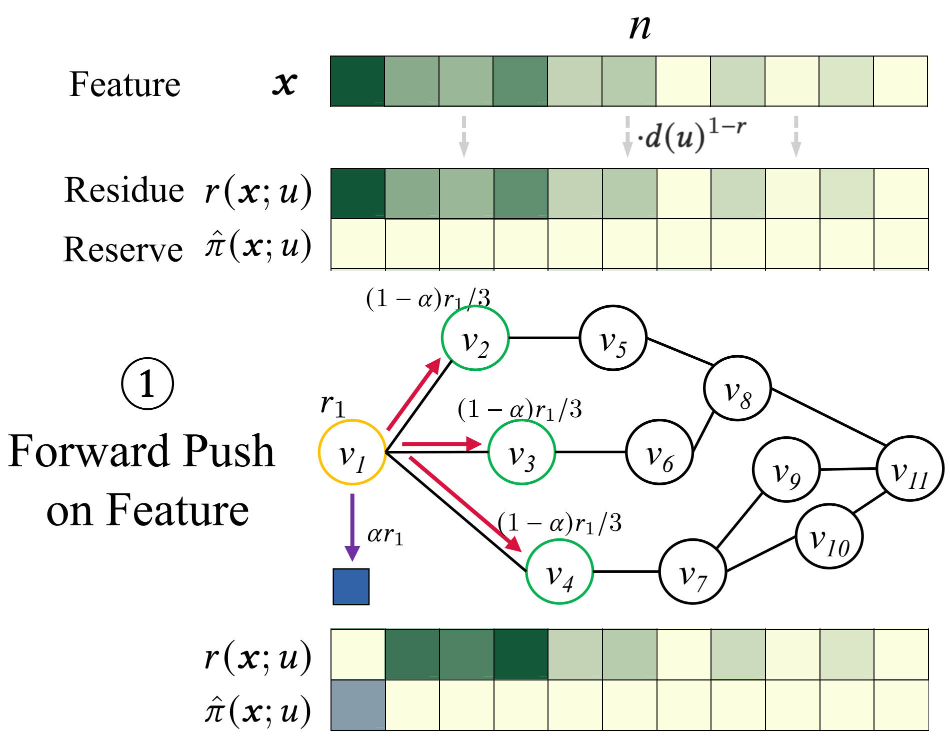
# Method: SCARA Framework

- Efficient propagation with feature-oriented algorithms
  - **FEATURE-PUSH**: *single* feature vector-based propagation
  - **FEATURE-REUSE**: reuse among *multiple* features



# Method: FEATURE-PUSH

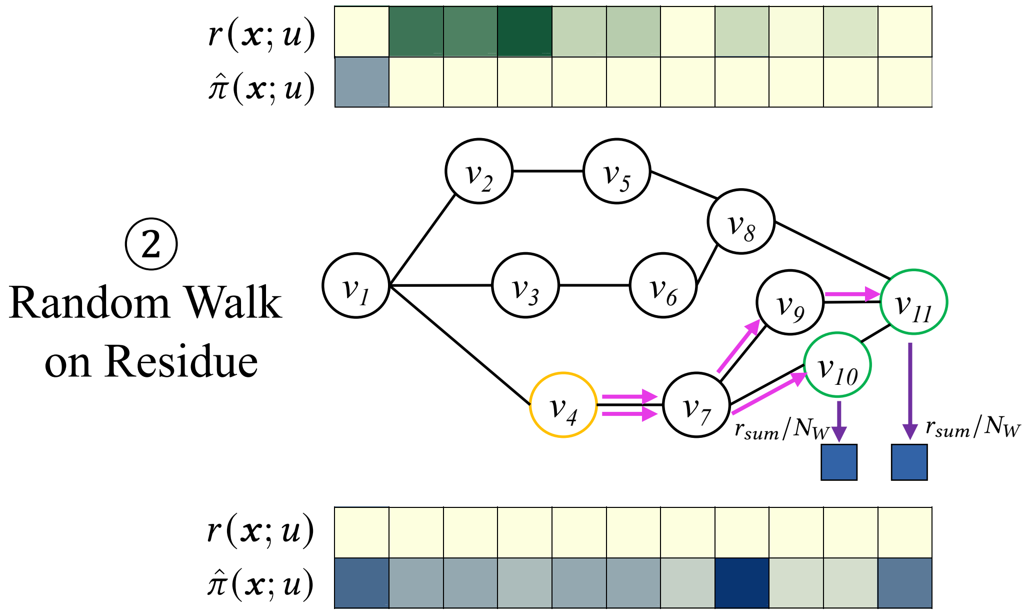
- INITIALIZATION: feature vector as residue variable
- STEP ①: Forward Push on Feature Value
  - Residue $\boldsymbol{r}$: values pending push
  - Reserve $\boldsymbol{\pi}$: underestimation of embedding value
  - Complexity: $O\left(\|\boldsymbol{x}\|_1 / r_{max}\right)$



# Method: FEATURE-PUSH

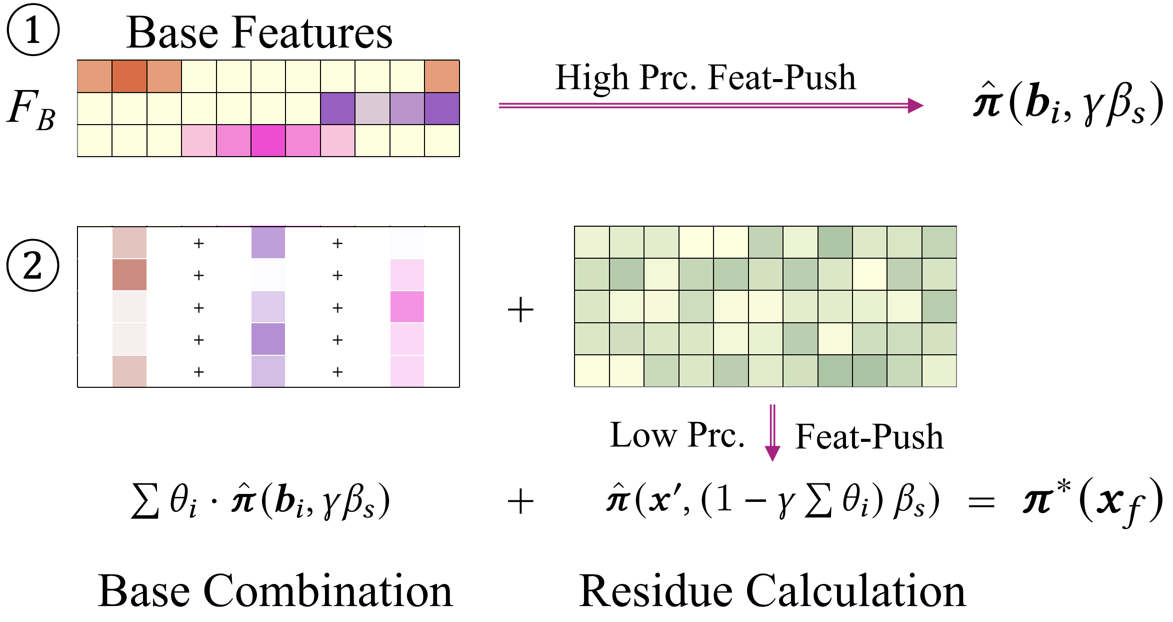
- STEP ②: Random Walk on Feature Residue
  - Increase reserves of end nodes based on walks
  - Precision guarantee
  - Complexity: $O\left(m \cdot r_{max}/\beta\right)$
- COMBINATION: Push Parameter $\beta$
  - Overall complexity: $O(\sqrt{\frac{m\|\boldsymbol{x}\|_1}{\beta}})$

$$\Longrightarrow O(\sqrt{m \log n}/\lambda)$$



# Method: FEATURE-REUSE

- Base Features
  - Select $F_B \ll F$ base features
  - High Prc. FEAT-PUSH with $\beta = \gamma\beta_s$
- Non-Base Features
  - Linear combination of base calculation results
  - Low Prc. FEAT-PUSH: sparse vectors thus faster
  - Complexity: $O\left(\sqrt{\frac{m(1-\theta_{sum})}{\beta_s(1-\gamma\theta_{sum})}}\right)$



# Experimental Evaluation

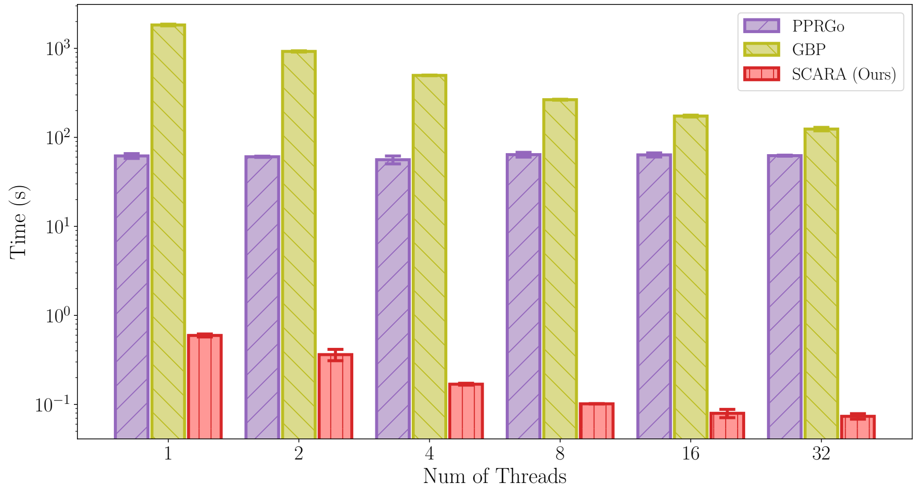
- **Time Efficiency**: 30-800× faster parallel precomputation, comparable or better training and inference clock time
- **Memory Efficiency**: Paper100M with 72GB without OOM
- **Effectiveness**: similar or better F1-score, fast convergence

| Dataset | Nodes $n$ | Edges $m$ | Features $F$ | Dataset | Nodes $n$ | Edges $m$ | Features $F$ |
|---|---|---|---|---|---|---|---|
| Reddit [15] | 232, 965 | 114, 615, 892 | 602 | Papers100M [16] | 111, 059, 956 | 1, 615, 685, 872 | 128 |

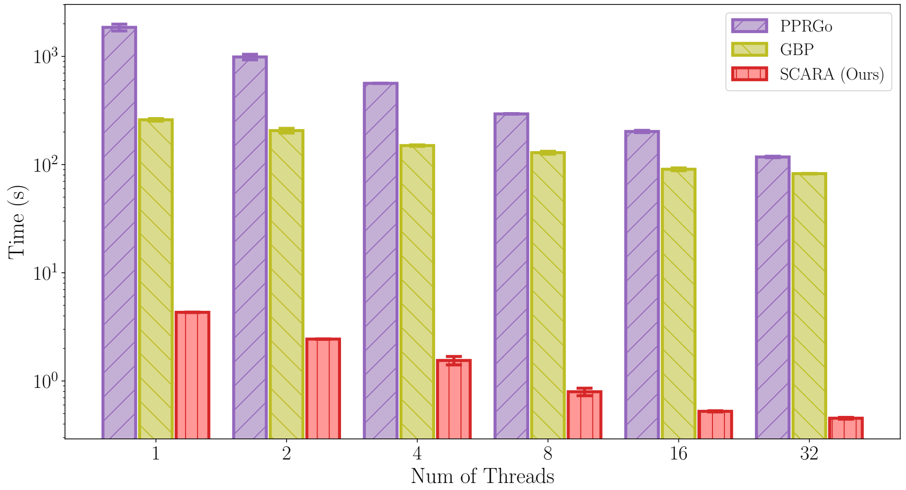
| Transductive | Reddit | | | | | | Papers100M | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Learn | ( Pre. + | Train) | Infer | Mem. | F1 | Learn | ( Pre. + | Train) | Infer | Mem. | F1 |
| GraphSAINT | 14.4 | ( – | 14.4) | 166.2 | 13.7 | 41.6 ±4.8 | – | – | – | – | OOM | – |
| GAS | 1151 | ( – | 1151) | **2.2** | 14.0 | 38.2 ±0.3 | – | – | – | – | OOM | – |
| PPRGo | 79.4 | (62.3 + | 17.1) | 29.1 | 9.4 | 41.5 ±2.3 | – | – | – | – | OOM | – |
| GBP | 138 | ( 124 + | 13.7) | 13.5 | 7.9 | 38.8 ±0.3 | – | – | – | – | OOM | – |
| **SCARA (ours)** | **13.9** | (0.07 + | 13.8) | 10.7 | **5.6** | **44.1 ±0.4** | **1346** | (12.7 + | 1333) | **4.7** | **71.4** | **35.7 ±0.9** |

# Experimental Evaluation

- **Time Efficiency**: 30-800× faster parallel precomputation, comparable or better training and inference clock time
- **Memory Efficiency**: Paper100M with 72GB without OOM
- **Effectiveness**: similar or better F1-score, fast convergence



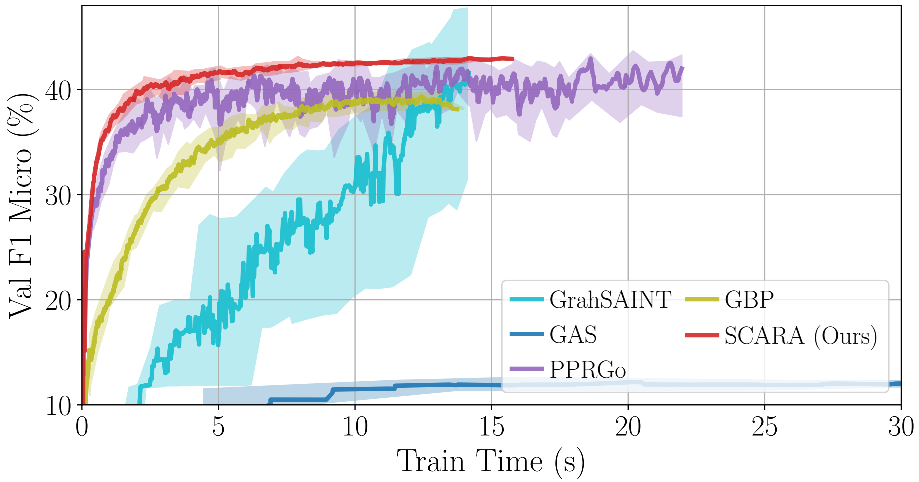

(A) Reddit



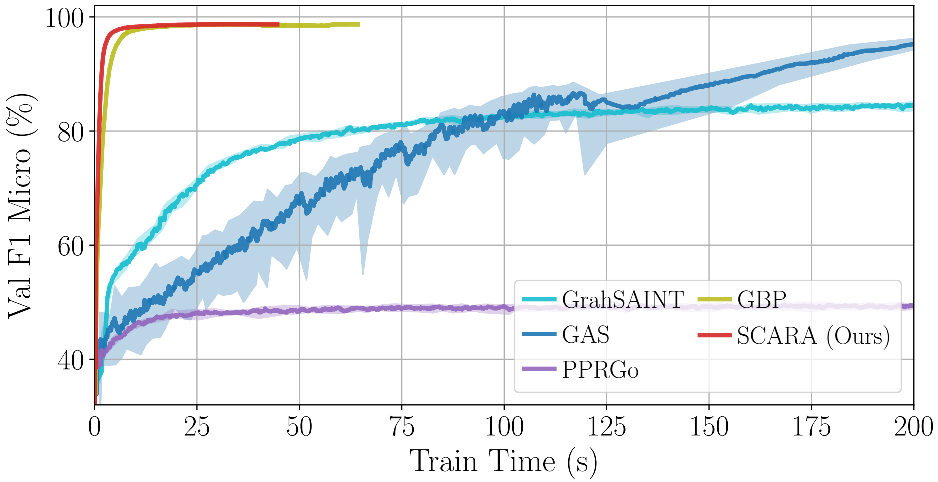

(B) Amazon

# Experimental Evaluation

- **Time Efficiency**: 30-800× faster parallel precomputation, comparable or better training and inference clock time
- **Memory Efficiency**: Paper100M with 72GB without OOM
- **Effectiveness**: similar or better F1-score, fast convergence




(a) Reddit




(b) PPI

# Summary

- **SCARA Framework**: Pre-Propagation Decoupled Model, optimized propagation with fast GPU batch training and inference
- **FEATURE-PUSH**: feature-oriented fast vector-based propagation, sub-linear precomputation complexity
- **FEATURE-REUSE**: efficient reuse among multiple features, further saves computation with guaranteed precision
- **Performance Evaluation**: up to 800× faster precomputation, able to process billion-scale Papers100M in 13 seconds

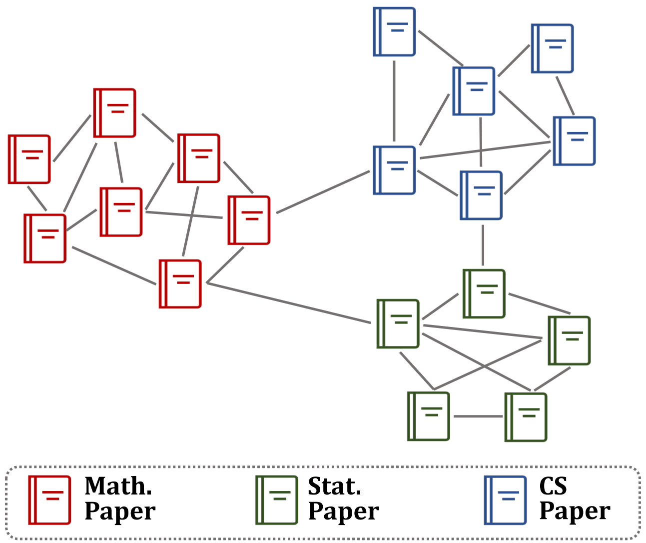
# $LD^2$: Scalable Heterophilous GNN with Decoupled Embedding

**Ningyi Liao**, Siqiang Luo, Xiang Li, Jieming Shi. "$LD^2$: Scalable Heterophilous Graph Neural Network with Decoupled Embedding". Under submission of *the 37th Conference on Neural Information Processing Systems*, 2023.

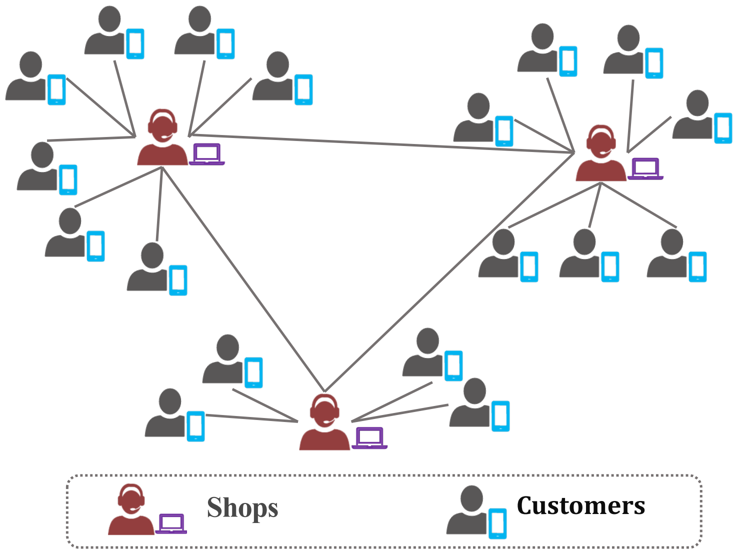
# Contents

# Graph Heterophily

- <u>Homophily</u>: connected nodes tend to be of *similar* classes
- <u>Heterophily</u>: connected nodes tend to be of *dissimilar* classes




Homophilous Graph: citation network

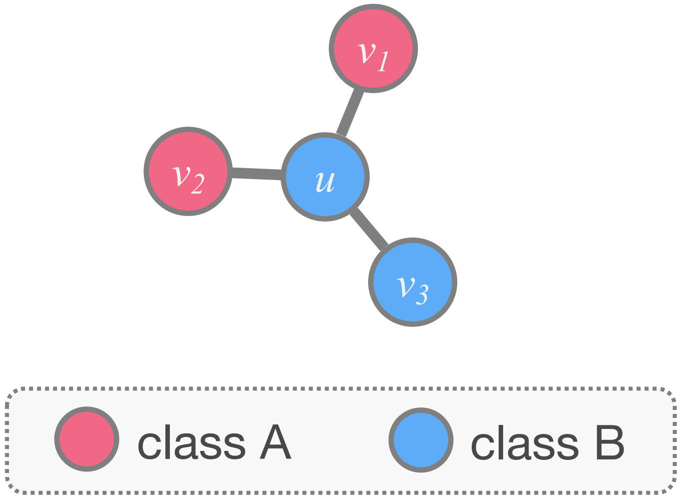


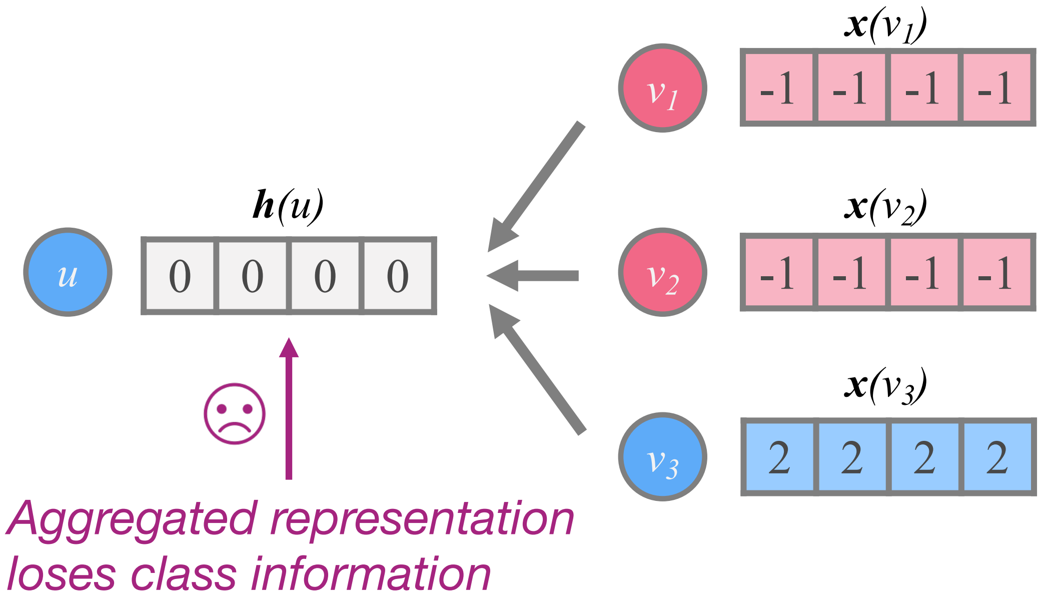

Heterophilous Graph: transaction network

[1] X Zheng et al. "Graph Neural Networks for Graphs with Heterophily: A Survey". *arXiv:2202.07082*. 2022.

# Graph Heterophily

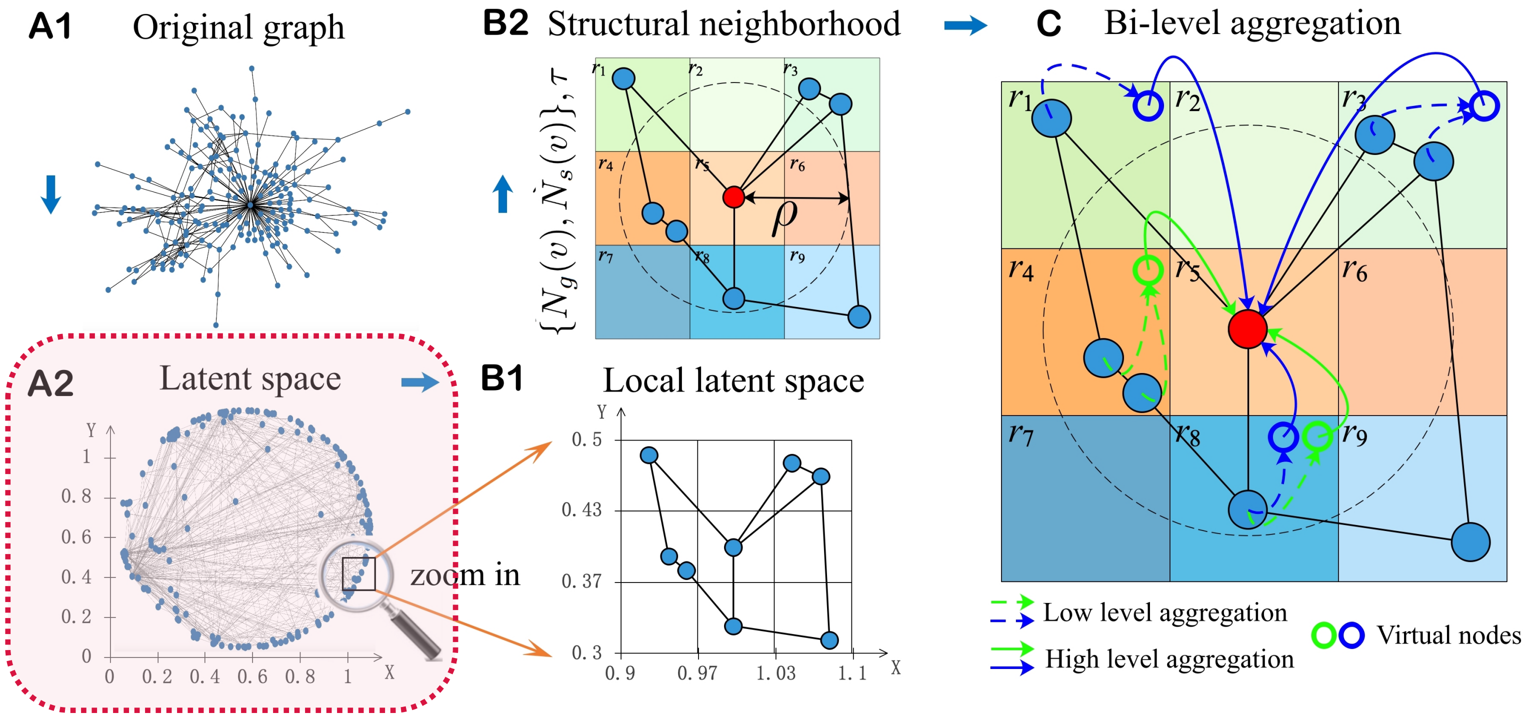
- Locality-based GNNs not suitable for heterophilous graphs



# Existing Work: Full-Graph Information

- Heterophilous GNNs usually rely on whole-graph information

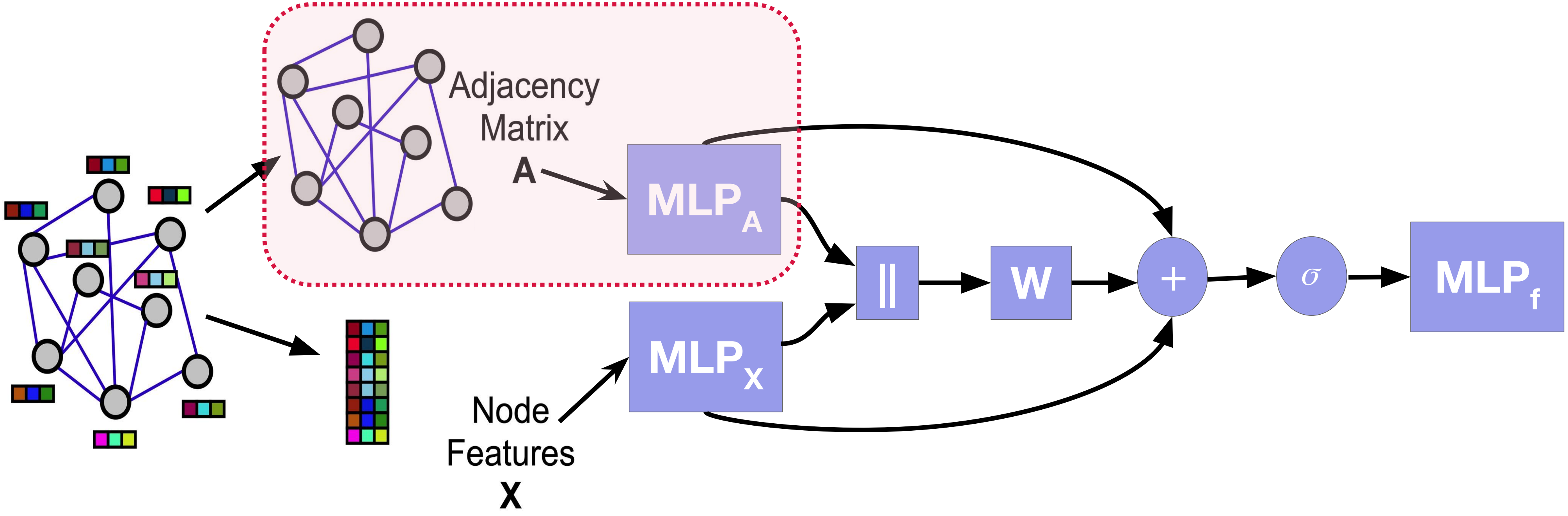



Geom-GCN: full graph embedding

[1] H Pei et al. "Geom-GCN: Geometric Graph Convolutional Networks". ICLR 2020.

# Existing Work: Full-Graph Information

- Heterophilous GNNs usually rely on <u>whole-graph information</u>




LINKX: entire graph adjacency matrix

[1] D Lim et al. "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods". NeurIPS 2021.

# Scalability Issue of Heterophilous GNNs

- Natural conflict: whole-graph processing not suitable for large-scale and minibatch tasks

Table 2.3: Precomputation, training, and inference time complexity of heterophilous GNNs.

| Model | Precomp. Time | Training Time | Inference Time |
|---|---|---|---|
| GPRGNN [39] | $O(m)$ | $O(IL_PmF + ILnF^2)$ | $O(L_PmF + LnF^2)$ |
| GCNJK [40] | – | $O(ILmF + ILnF^2)$ | $O(LmF + LnF^2)$ |
| MixHop [41] | – | $O(IL_PLmF + ILnF^2)$ | $O(L_PLmF + LnF^2)$ |
| LINKX [42] | – | $O(ImF + ILnF^2)$ | $O(mF + LnF^2)$ |
| **LD$^2$ (ours)** | $O(L_PmF)$ | $O(ILnF^2)$ | $O(LnF^2)$ |

☹ *Terms that not scalable to large $m$ and $n$*

Table 2.4: Precomputation, training, and inference memory complexity of heterophilous GNNs.

| Model | Precomp. Mem. | Training Mem. | Inference Mem. |
|---|---|---|---|
| GPRGNN [39] | $O(m)$ | $O(LnF + LF^2 + m)$ | $O(LnF + LF^2 + m)$ |
| GCNJK [40] | – | $O(L_CnF + L_CF^2)$ | $O(L_CnF + L_CF^2)$ |
| MixHop [41] | – | $O(CLnF + CLF^2)$ | $O(CLnF + CLF^2)$ |
| LINKX [42] | – | $O(L_Cn_bF + L_CF^2 + nF)$ | $O(L_Cn_bF + L_CF^2 + nF)$ |
| **LD$^2$ (ours)** | $O(CnF)$ | $O(L_Cn_bF + L_CF^2)$ | $O(L_Cn_bF + L_CF^2)$ |

☹ *Terms that not suitable for minibatch*

# Our Model: LD²

- Precomputation:

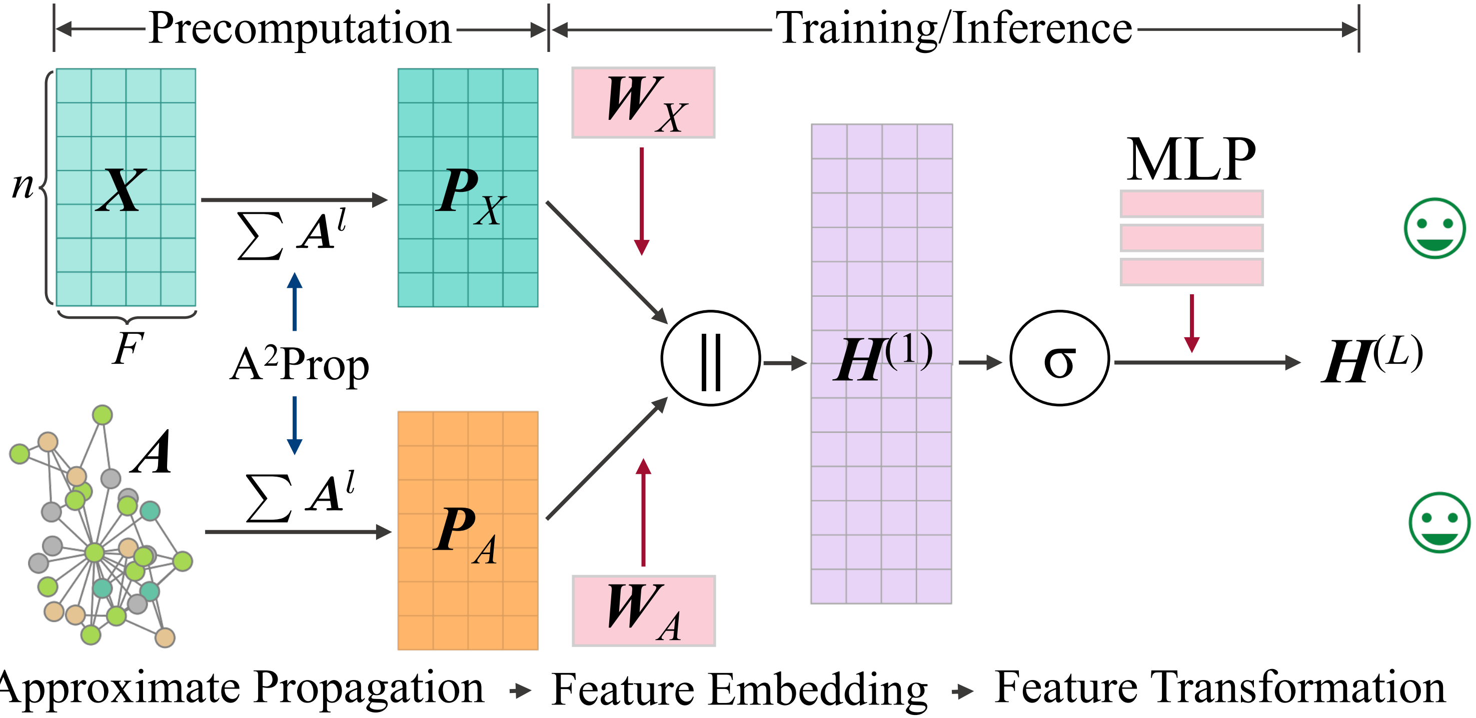
$$\boldsymbol{P}_A, \boldsymbol{P}_X = \mathrm{A}^2\mathrm{Prop}(\boldsymbol{A}, \boldsymbol{X})$$

- Feature Transformation:

$$\boldsymbol{H}^{(L)} = \mathrm{MLP}(\boldsymbol{P}_A \boldsymbol{W}_A \| \boldsymbol{P}_X \boldsymbol{W}_X)$$

Long-Distance Feature Embedding

Low-Dimension Adj Embedding

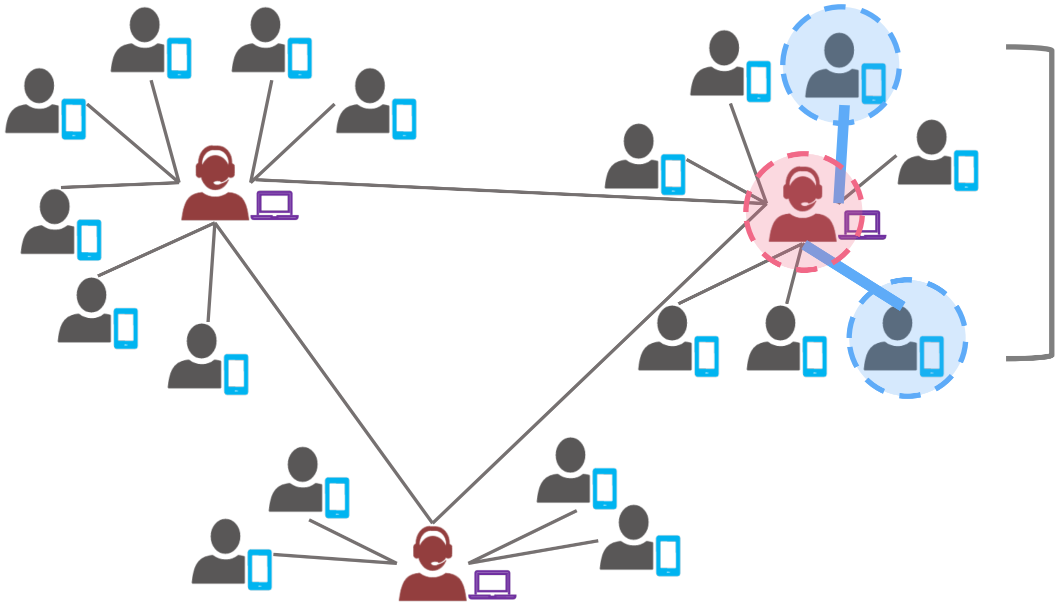



*Linear to $m$ and $n$*

*Decoupled simple minibatching*

# Method: Adjacency Embedding

- 2-hop neighbors are proved to be homophily-dominant regardless of 1-hop neighbor distribution



2-hop nodes are usually similar

[1] J Zhu et al. “Beyond homophily in graph neural networks: Current limitations and effective designs.”. NeurIPS 2020.

# Method: Adjacency Embedding

- Low-Dimension 2-hop adj decomposition as embedding:

$$\underbrace{\boldsymbol{P}_A}_{\substack{\text{Adjacency Embedding} \\ n \times F}} \cdot \boldsymbol{P}_A^\top \approx \underbrace{\boldsymbol{A}^2}_{\substack{\text{2-hop Adjacency Matrix} \\ n \times n}}$$

- Calculation:

$$\boldsymbol{P}_A^{(0)} = \underbrace{\mathcal{N}}_{\text{Gaussian Matrix}}$$

$$\boldsymbol{P}_A^{(i+1)} = \phi(\boldsymbol{A}^2 \cdot \boldsymbol{P}_A^{(i)})$$

Power Iteration

$$\Leftrightarrow$$

$$\boldsymbol{P}_A = \underbrace{\phi}_{\text{Orthogonalize + Normalize}}\left(\boldsymbol{A}^2 \phi\left(\boldsymbol{A}^2 \cdots \phi\left(\boldsymbol{A}^2 \mathcal{N}\right)\right)\right)$$

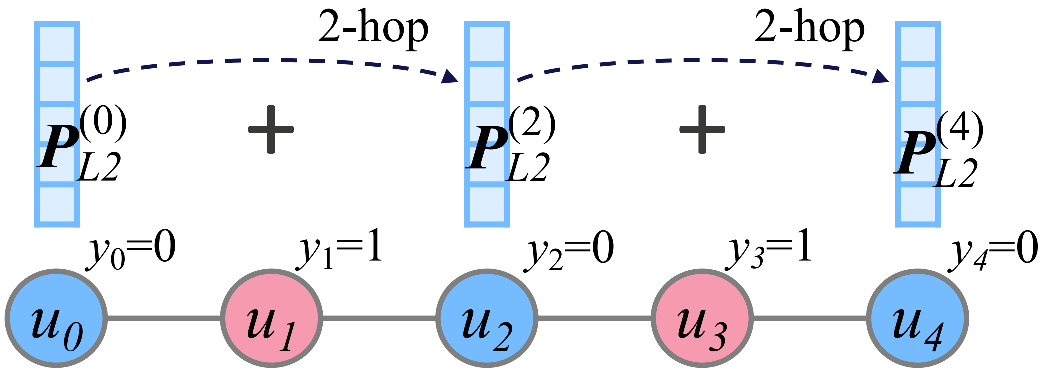
Iterative Adj Multiplication

# Method: Feature Embedding

- <u>L</u>ong-<u>D</u>istance generalized graph propagation

CHANNEL ①: *Constant* 2-hop Adjacency Propagation

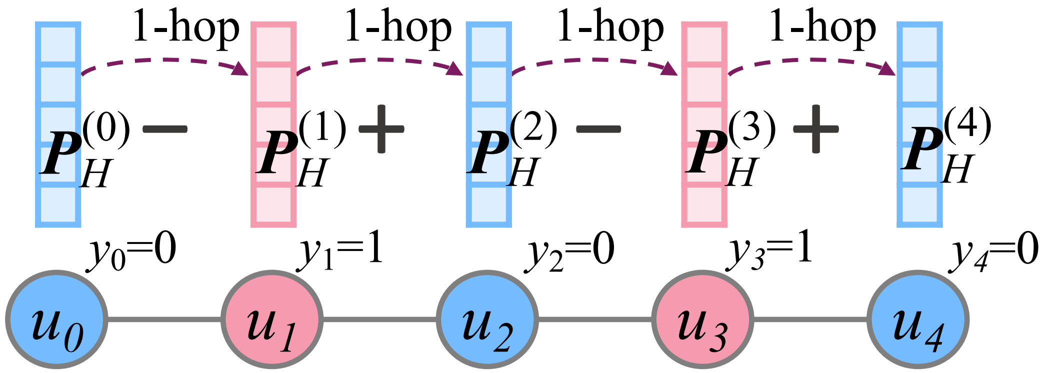
$$\boldsymbol{P}_{X,L2} = \sum_{l=1}^{L} \bar{\boldsymbol{A}}^{2l} \cdot \boldsymbol{X}$$




CHANNEL ②: *Inverse* 1-hop Laplacian Propagation

Laplacian Matrix $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{A}$

$$\boldsymbol{P}_{X,H} = \sum_{l=1}^{L} \tilde{\boldsymbol{L}}^{l} \cdot \boldsymbol{X}$$

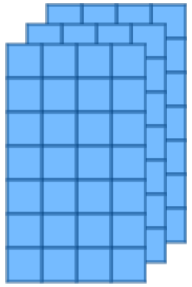


# Method: Embedding Calculation

- Multi-channel embedding, one-time computation:

| Channel | |
|---|---|
| Adjacency | $\boldsymbol{P}_A = \phi\left(\boldsymbol{A}^{2L} \cdot \boldsymbol{\mathcal{N}}\right)$ |
| Feature – 2-hop Constant | $\boldsymbol{P}_{X,L2} = \sum_{l=1}^{L} \bar{\boldsymbol{A}}^{2l} \cdot \boldsymbol{X}$ |
| Feature – 1-hop Inverse | $\boldsymbol{P}_{X,H} = \sum_{l=1}^{L} \tilde{\boldsymbol{L}}^{l} \cdot \boldsymbol{X}$ |

$\xrightarrow[O(LmF)]{\text{Precomputation}}$



Embedding Input

# Experimental Evaluation

- **Effectiveness**:
  - Top 1 accuracy on 6/8 graphs
  - No accuracy drop for minibatch

| Dataset | squirrel | genius | penn94 | arxiv-year | twitch-gamers | pokec | snap-patents | wiki |
|---|---|---|---|---|---|---|---|---|
| Nodes $n$ | 5,201 | 421,858 | 41,536 | 169,343 | 168,114 | 1,632,803 | **2,738,035** | 1,770,981 |
| Edges $m$ | 401,907 | 1,344,722 | 1,403,756 | 1,327,142 | 6,965,671 | 23,934,767 | 16,705,984 | **244,278,050** |
| $F$ / $N_c$ | 2,089 / 5 | 12 / 2 | 4,814 / 2 | 128 / 5 | 7 / 2 | 65 / 2 | 269 / 5 | 600 / 5 |
| GCNJK-GS | 27.63 ±4.72 | 80.65 ±0.07 | 65.91 ±0.16 | 48.26 ±0.64 | 59.91 ±0.42 | 59.38 ±0.21 | 33.64 ±0.05 | 42.95 ±0.39 |
| MixHop-GS | 33.24 ±2.44 | 80.63 ±0.04 | 75.00 ±0.37 | 49.26 ±0.16 | 61.80 ±0.00 | 64.02 ±0.02 | 34.73 ±0.15 | 45.52 ±0.11 |
| LINKX | 60.14 ±0.92 | 82.51 ±0.10 | **78.63** ±0.25 | **50.44** ±0.30 | 64.15 ±0.18 | 68.64 ±0.65 | 52.69 ±0.05 | 50.59 ±0.12 |
| LD$^2$ (ours) | **66.87** ±0.02 | **85.31** ±0.06 | 75.52 ±0.10 | 50.29 ±0.11 | **64.33** ±0.19 | **74.93** ±0.10 | **58.58** ±0.34 | **52.91** ±0.16 |

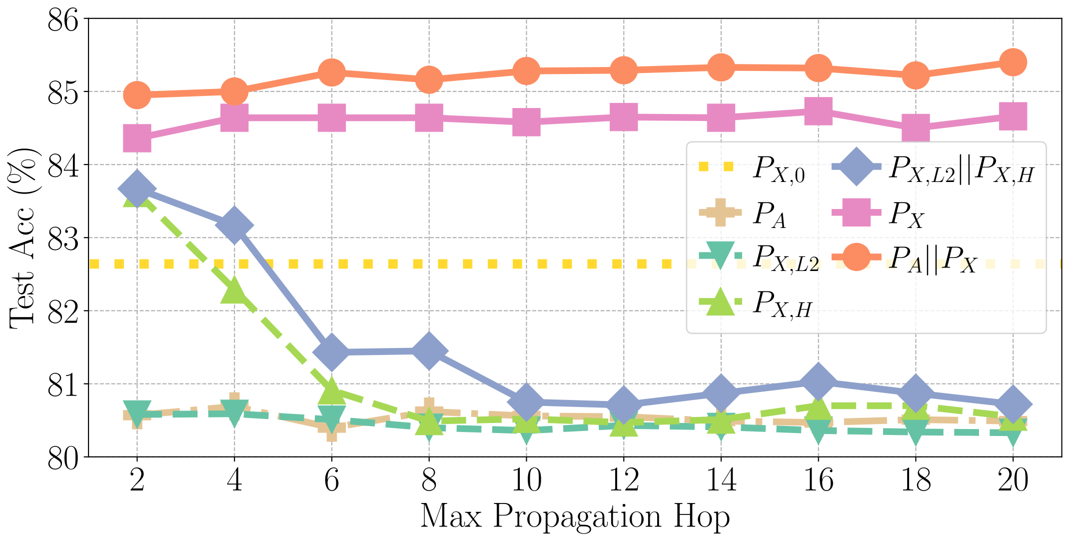
# Experimental Evaluation

- **Efficiency**:
  - 3-15× faster minibatch training, significantly fast inference
  - Up to 5× lower memory for large graphs

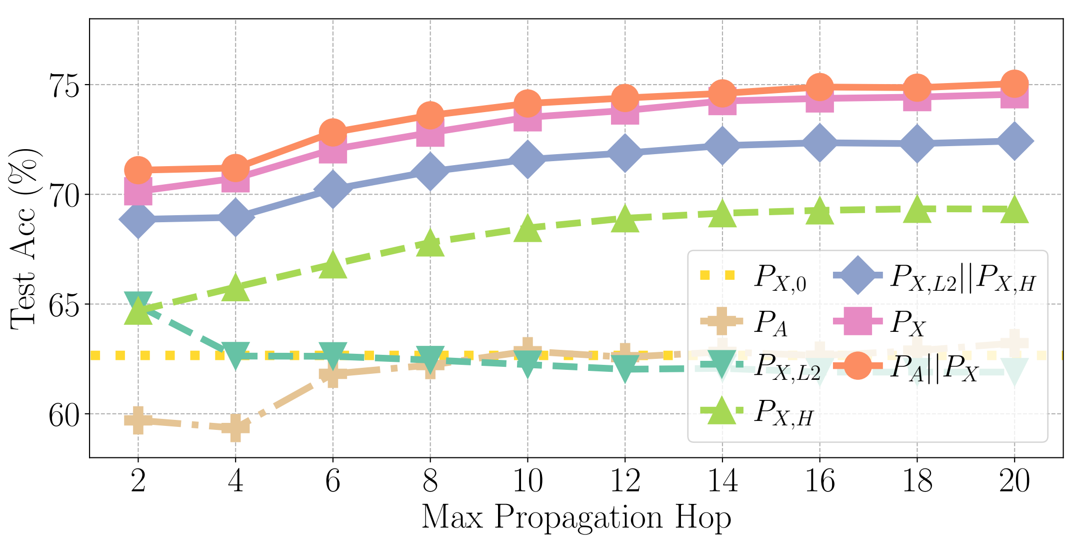
| Dataset | twitch-gamers | | | pokec | | | snap-patents | | | wiki | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Learn | Infer | Mem. | Learn | Infer | Mem. | Learn | Infer | Mem. | Learn | Infer | Mem. |
| MLP | 6.36 | 0.02 | 0.61 | 47.86 | 0.11 | 13.77 | 27.39 | 0.28 | 9.33 | 133.55 | 0.62 | 18.15 |
| PPRGo | 10.46+15.88 | 0.41 | 9.64 | 121.95+56.11 | 2.69 | 3.82 | | (>12h) | | | (>12h) | |
| SGC | 0.09+0.74 | 0.01 | 0.28 | 1.05+8.08 | 0.01 | 0.28 | 4.94+23.54 | 0.01 | 0.42 | 12.66+7.98 | 0.01 | 0.52 |
| GCNJK-GS | 71.48 | 0.02* | 7.33 | 27.33 | 0.09* | 9.03 | 19.02 | 0.23* | 9.21 | 95.52 | 0.69* | 16.36 |
| MixHop-GS | 52.12 | 0.01* | 1.49 | 71.35 | 0.03* | 12.91 | 45.24 | 0.16* | 19.58 | 84.22 | 0.23* | 16.28 |
| LINKX | 10.99 | 0.19 | 2.35 | 28.77 | 0.33 | 9.03 | 39.80 | 0.22 | 21.53 | 180.71 | 1.14 | 14.53 |
| **LD$^2$ (ours)** | 0.85+**1.96** | **0.01** | **1.44** | 17.95+**6.18** | **0.01** | **3.82** | 31.32+**6.96** | **0.02** | **3.96** | 28.12+**6.50** | **0.01** | **4.47** |

# Experimental Evaluation

- **Ablation Study**:
  - Combination of channels is effective for different graphs




genius




pokec

# Summary

- **$LD^2$ Framework**: Pre-Propagation Decoupled Heterophilous GNN, optimized minibatch training
- **Low Dimension Adjacency Embedding**: embed full graph topology in low dimensional matrix representation
- **Long Distance Feature Embeddings**: multi-channel node features by different graph propagation schemes
- **Performance Evaluation**: 3–15× faster minibatch training and inference, up to 5× smaller memory footprint

# Contents

# Conclusion

- CONTRIBUTION ①: **SCARA**
  - Scalable GNN with decoupled efficient graph propagation and feature-oriented optimizations
  - Sub-linear complexity, fast precomputation, 1.6B graph in 13 sec
- CONTRIBUTION ②: **LD$^2$**
  - Scalable heterophilous GNN with decoupled multi-channel topology and feature embeddings
  - Minibatch ability, simplified learning, 240M graph in 40 sec

# Future Works

- **Broader Range of Models**:
  - Extend propagation schemes: iterative-, post-propagation
  - Apply on more architectures: GAT, Graph Transformer
- **Variants of Graph Data**:
  - Heterogeneous Graph: dissimilar edges and propagations
  - Dynamic Graph: nodes and edges change with time
- **Benchmarking GNN Scalability**:
  - Evaluate performance of different scalable GNN designs
  - Explore efficacy-efficiency tradeoff

# Contents

# THANK YOU