tu dortmund university

SFB 876 Providing Information
by Resource-Constrained Data Analysis

Fast Graph Representation Learning with PyTorch geometric

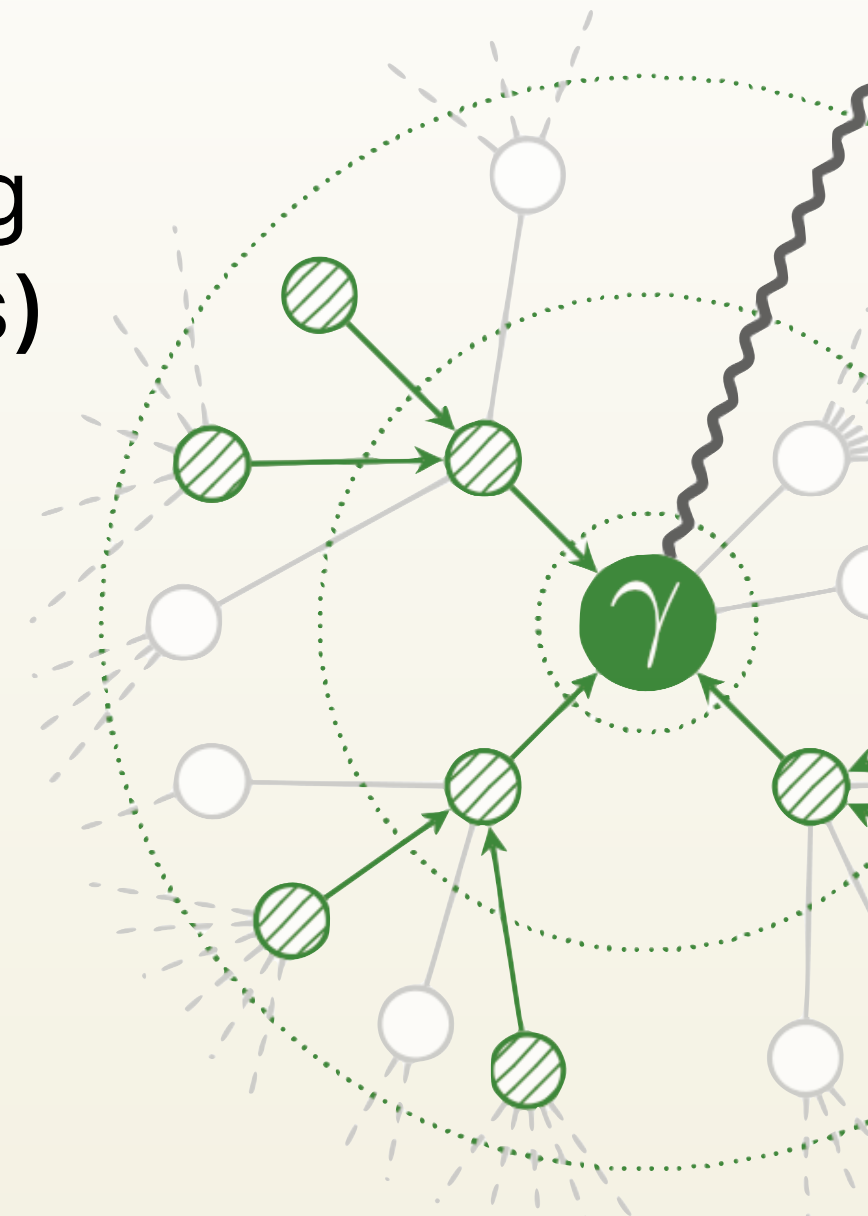
Matthias Fey & Jan Eric Lenssen

{matthias.fey,janeric.lenssen}@udo.edu

Introduction

PyTorch Geometric (PyG) is a **PyTorch** library for deep learning on **graphs, point clouds** and **manifolds**

- simplifies implementing and working with **Graph Neural Networks (GNNs)**
- bundles **fast implementations** from published papers
- tries to be **easily comprehensible** and **non-magical**

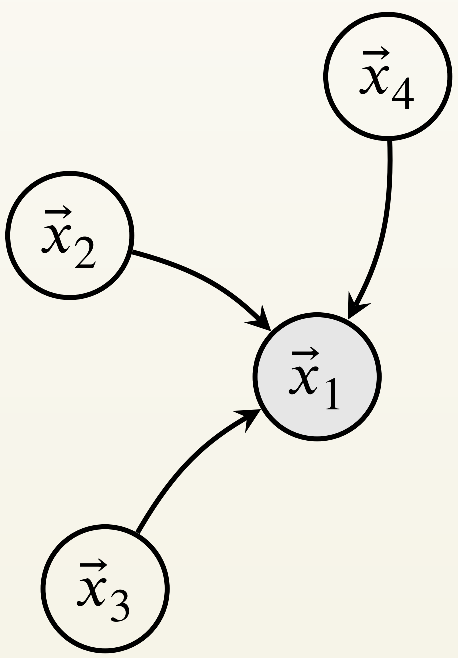


Graph Neural Networks

Given a *sparse* graph $\mathcal{G} = (\mathbf{X}, (\mathbf{I}, \mathbf{E}))$ with

$$\mathbf{X} = \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vec{x}_3 \\ \vec{x}_4 \end{bmatrix}$$

- node features $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$
- edge indices $\mathbf{I} \in \{1, \ldots, N\}^{2 \times |\mathcal{E}|}$
- *optional* edge features $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times D}$



$$\mathbf{I} = \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}^{\top}$$

Message Passing Scheme

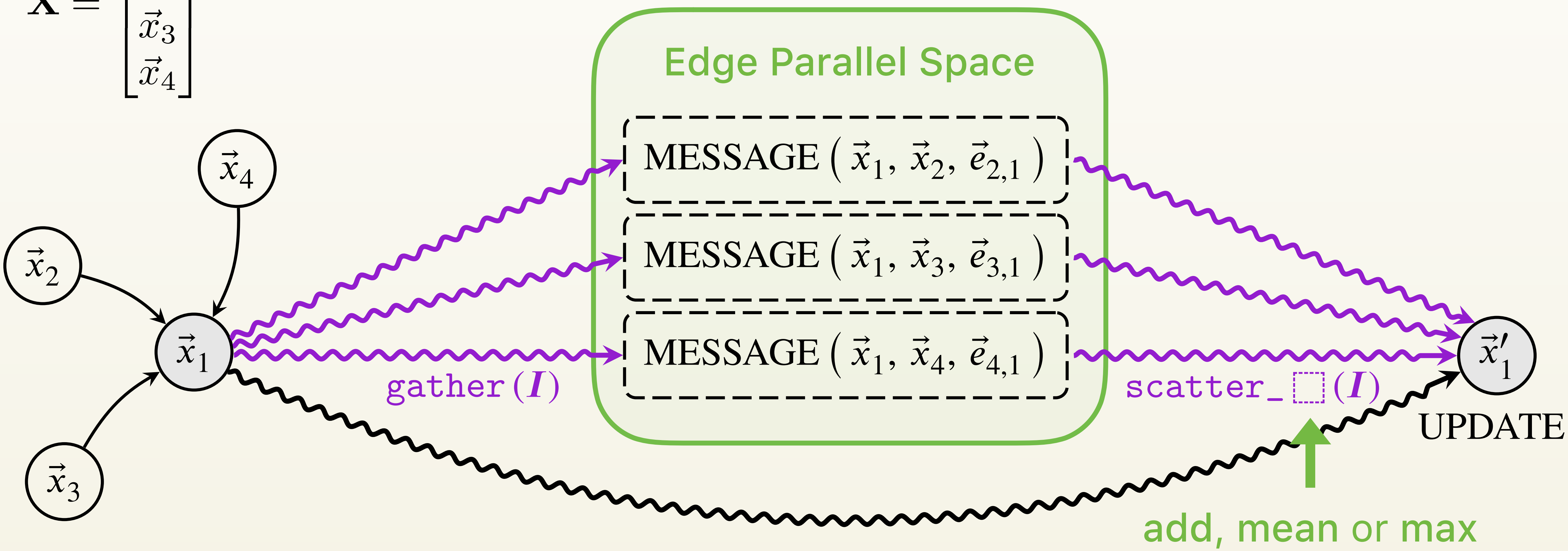
permutation-invariant aggregation operator

$$\vec{x}'_i = \text{UPDATE}\left(\vec{x}_i, \; \square_{j \in \mathcal{N}(i)} \; \text{MESSAGE}\left(\vec{x}_i, \; \vec{x}_j, \; \vec{e}_{j,i}\right) \right)$$

Neighborhood set $\mathcal{N}(i) = \{j : (j, i) \in \mathcal{E}\}$

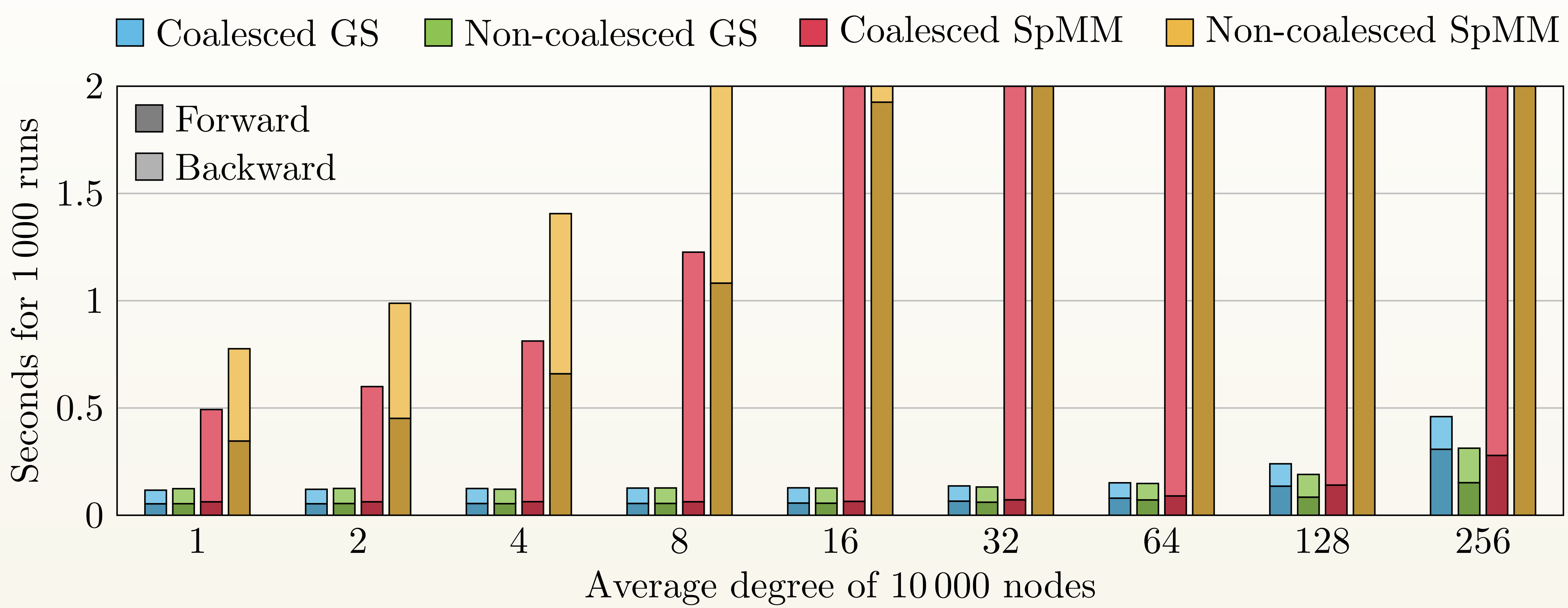
Flexible implementation via **Gather/Scatter operations**

$$\mathbf{X} = \begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vec{x}_3 \\ \vec{x}_4 \end{bmatrix}$$



$$\mathbf{I} = \begin{bmatrix} 2 & 1 \\ 3 & 1 \\ 4 & 1 \end{bmatrix}^\top$$

Graph Neural Networks



Gather/Scatter (GS) VS **Sparse-Matrix Multiplication (SpMM)**

Gather/Scatter (GS)

- ✓ input does not need to be coalesced
- ✓ can integrate central node and multi-dimensional edge information
- ✗ begins to struggle on dense graphs
- ✗ non-deterministic by nature on GPU

Sparse-Matrix Multiplication (SpMM)

- ✗ input needs to be coalesced *(backward pass is inherently slow)*
- ✗ *can only integrate node information*
- ✓ *efficient memory usage*

Message Passing interface

```python
class MyOwnConv(MessagePassing):
    def __init__(self):
        super(MyOwnConv, self).__init__(aggr='add')

    def forward(self, x, edge_index, e):
        return self.propagate(edge_index, x=x, e=e)

    def message(self, x_j, x_i, e):
        return x_j * e
```

add, mean or max aggregation

pass every-thing needed for propagation

Node features get **automatically mapped** to source (_j) and target (_i) nodes

Supports bipartite graphs!

Implemented Operators and Models

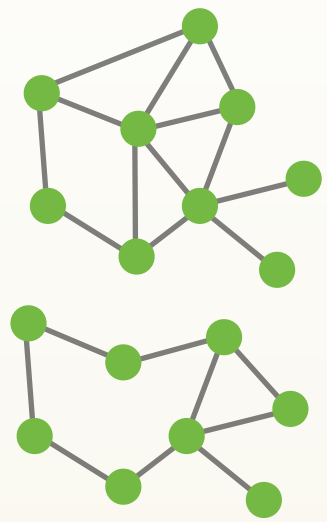
Cheby Defferrard et al. (2016)	**GCN** Kipf & Welling (2017)	**GAE** Kipf & Welling (2016)	**SAGE** Hamilton et al. (2017)
PointNet Qi et al. (2017)	**MoNet** Monti et al. (2017)	**MPNN** Gilmer et al. (2017)	**GAT** Veličković et al. (2018)
SplineCNN Fey et al. (2018)	**AGNN** Thekumparampil et al. (2018)	**EdgeCNN** Wang et al. (2018)	**JK** Xu et al. (2018)
S-GCN Derr et al. (2018)	**R-GCN** Schlichtkrull et al. (2018)	**PointCNN** Li et al. (2018)	**SGC** Wu et al. (2019)
ARMA Bianchi et al. (2019)	**APPNP** Klicpera et al. (2019)	**GIN** Xu et al. (2019)	**DGI** Veličković et al. (2019)

Presented here at ICLR - check them out!

Mini-Batching and Readout Functions

$\mathcal{G}_1 = (\mathbf{X}_1, \mathbf{A}_1)$

$\mathcal{G}_2 = (\mathbf{X}_2, \mathbf{A}_2)$



$$\mathrm{GNN}\left(\begin{bmatrix} \mathbf{A}_1 & \\ & \mathbf{A}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \end{bmatrix} \right) = \begin{bmatrix} \mathbf{X}'_1 \\ \mathbf{X}'_2 \end{bmatrix}$$

assignment vector $\begin{bmatrix} 0, \ldots, 0 & 1, \ldots, 1 \end{bmatrix}$

readout function

- *no GNN modifications needed*
- *no memory/computation overhead*
- *supports examples of different size*

Global Add/Mean/Max

Set2Set
Vinyals et al. (2016)

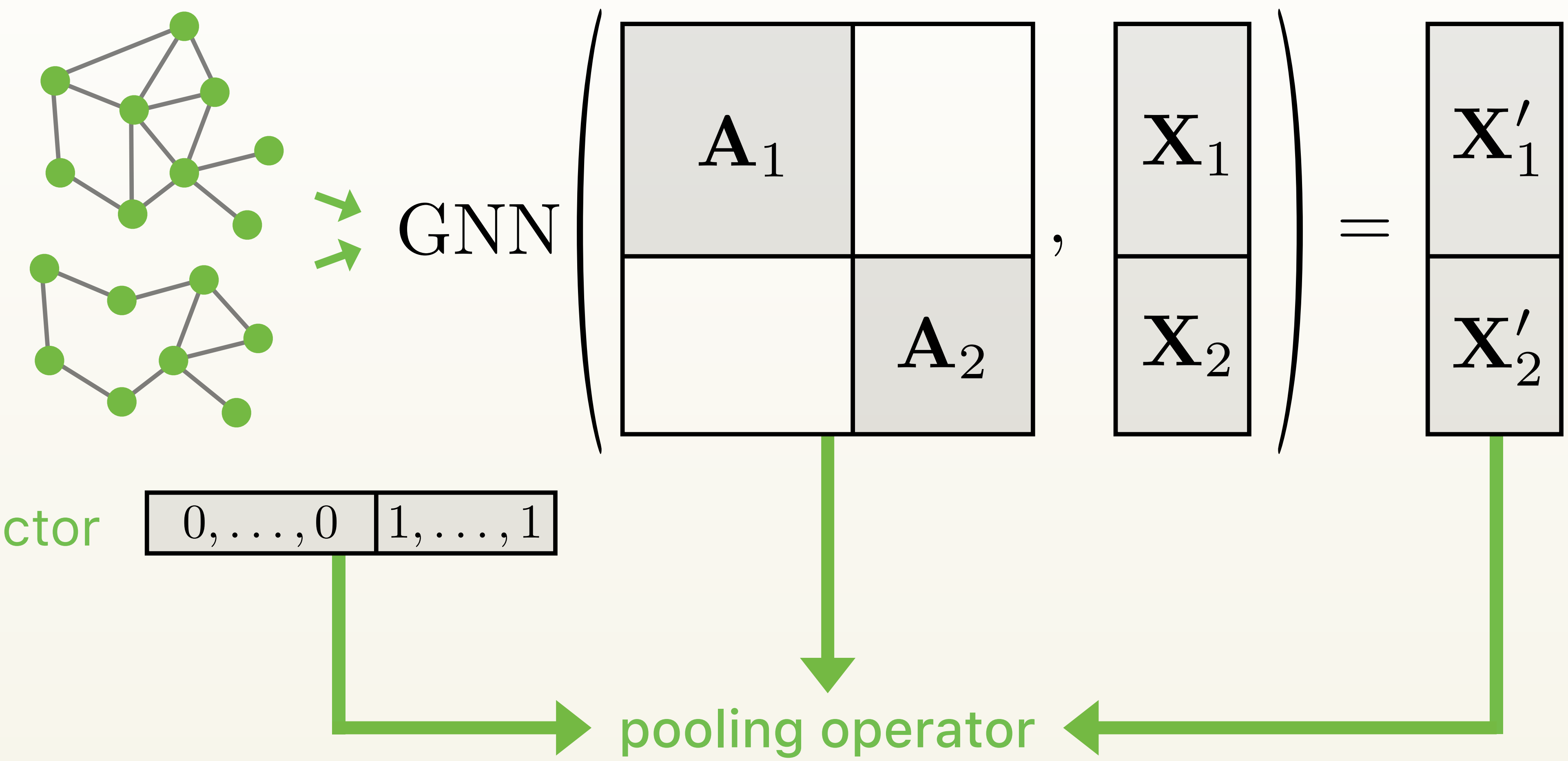
SortPool
Zhang et al. (2018)

GlobalAttention
Li et al. (2016)

Mini-Batching and Pooling Operators

$$\mathcal{G}_1 = (\mathbf{X}_1, \mathbf{A}_1)$$

$$\mathcal{G}_2 = (\mathbf{X}_2, \mathbf{A}_2)$$



Graclus	**Voxel**	**FPS**	**TopK**	**DiffPool**
Defferrard et al. (2016)	Simonovsky et al. (2017)	Qi et al. (2017)	Gao et al. (2018)	Ying et al. (2018)

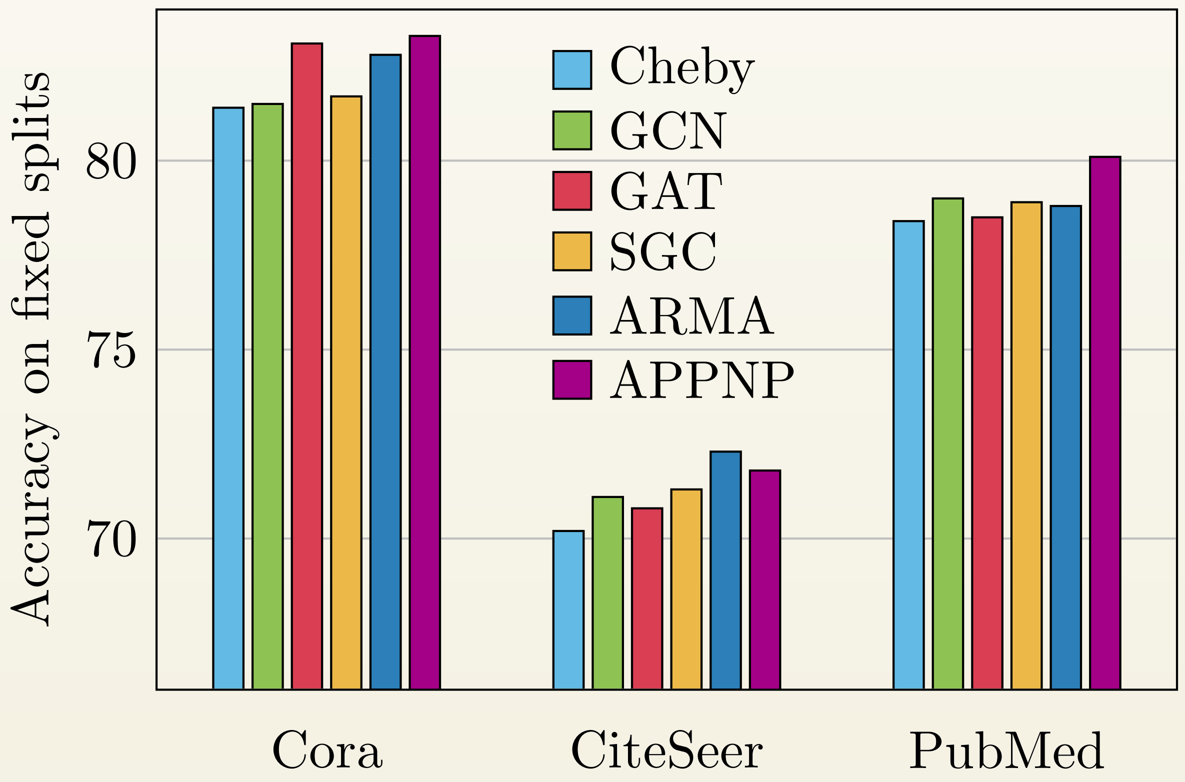
deterministic *differentiable*

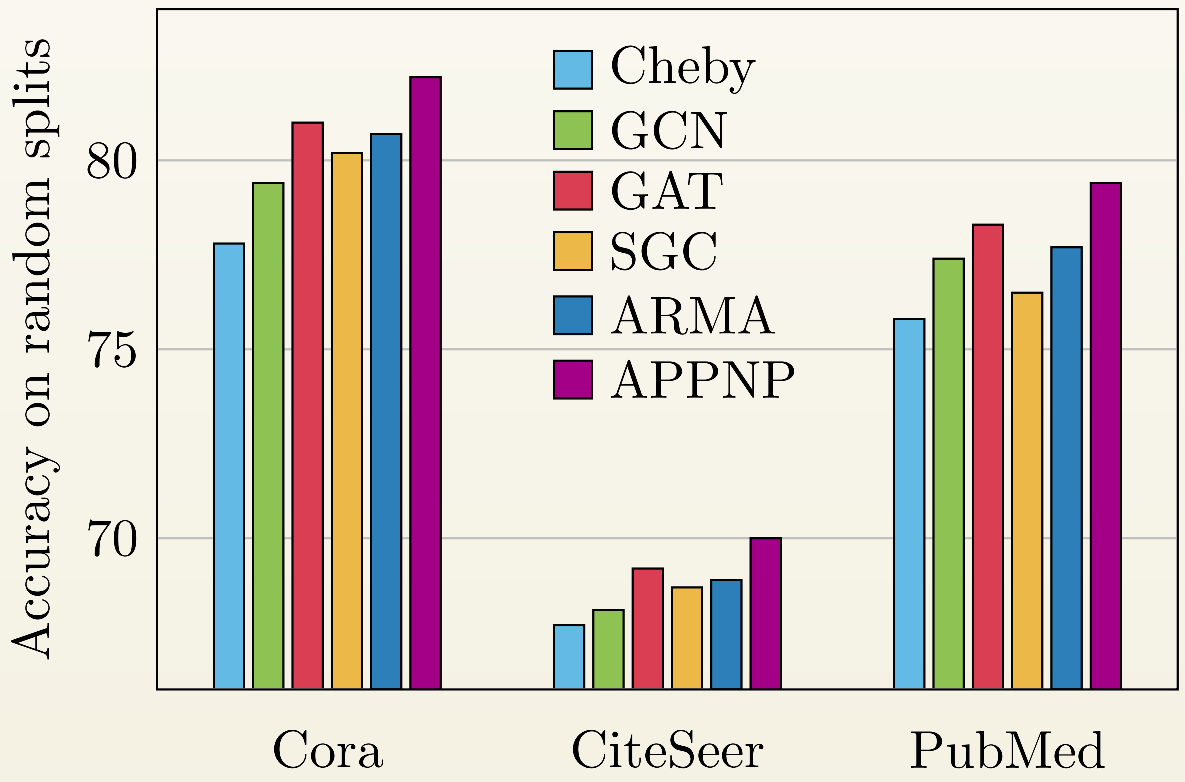
Demo

Experimental Evaluation

Easy-to-use benchmark scripts for evaluating new **research ideas**

Evaluation on **fixed** *and* **random splits**

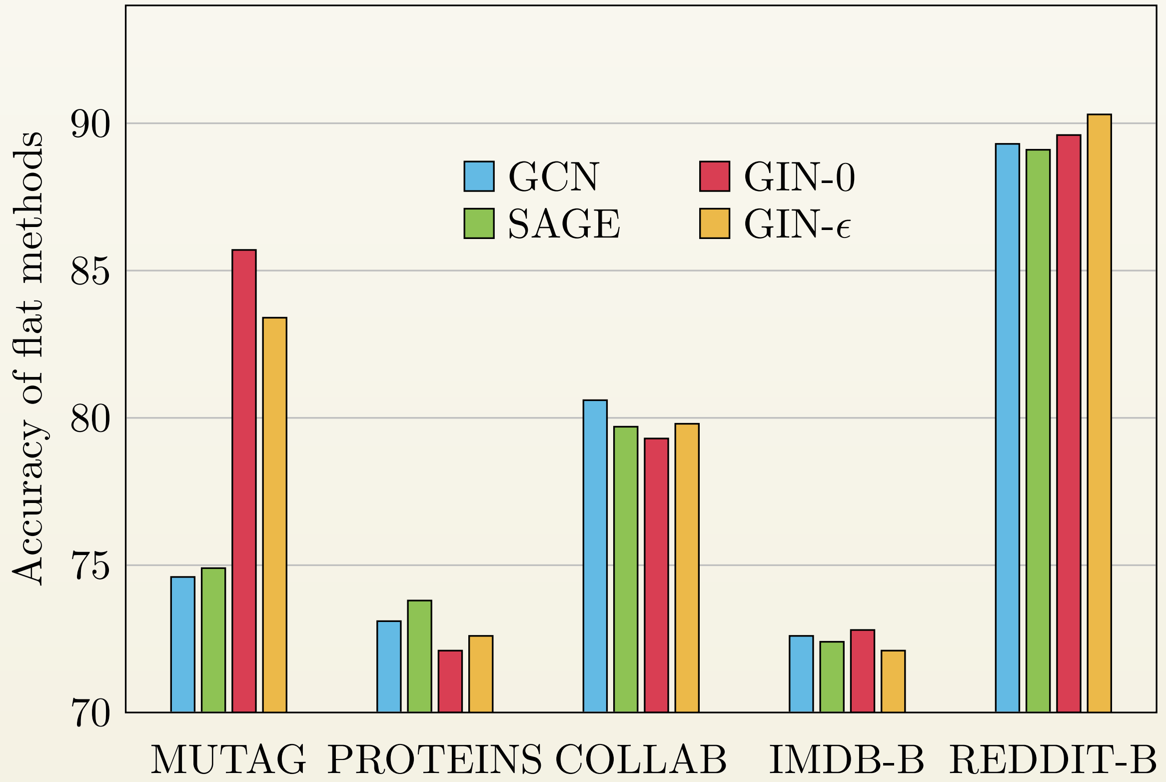


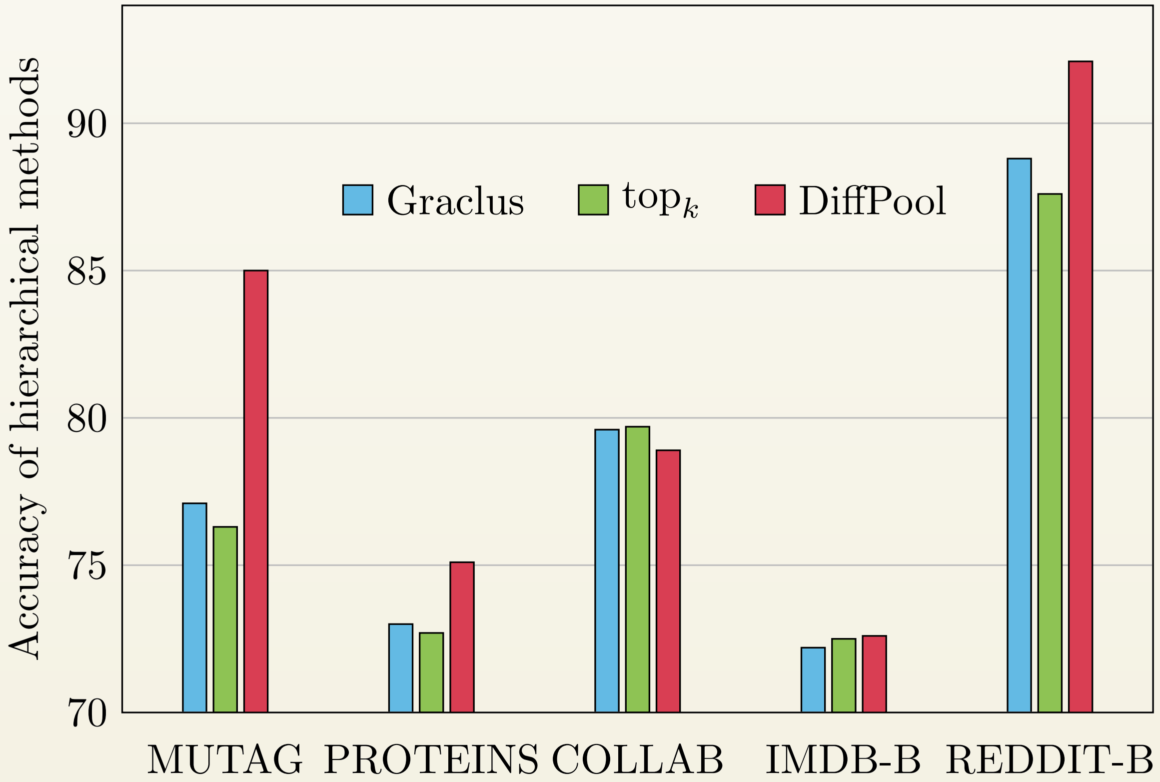


Experimental Evaluation

Easy-to-use benchmark scripts for evaluating new **research ideas**

Evaluation based on **cross validation**
with a **randomly sampled validation set**





Experimental Evaluation

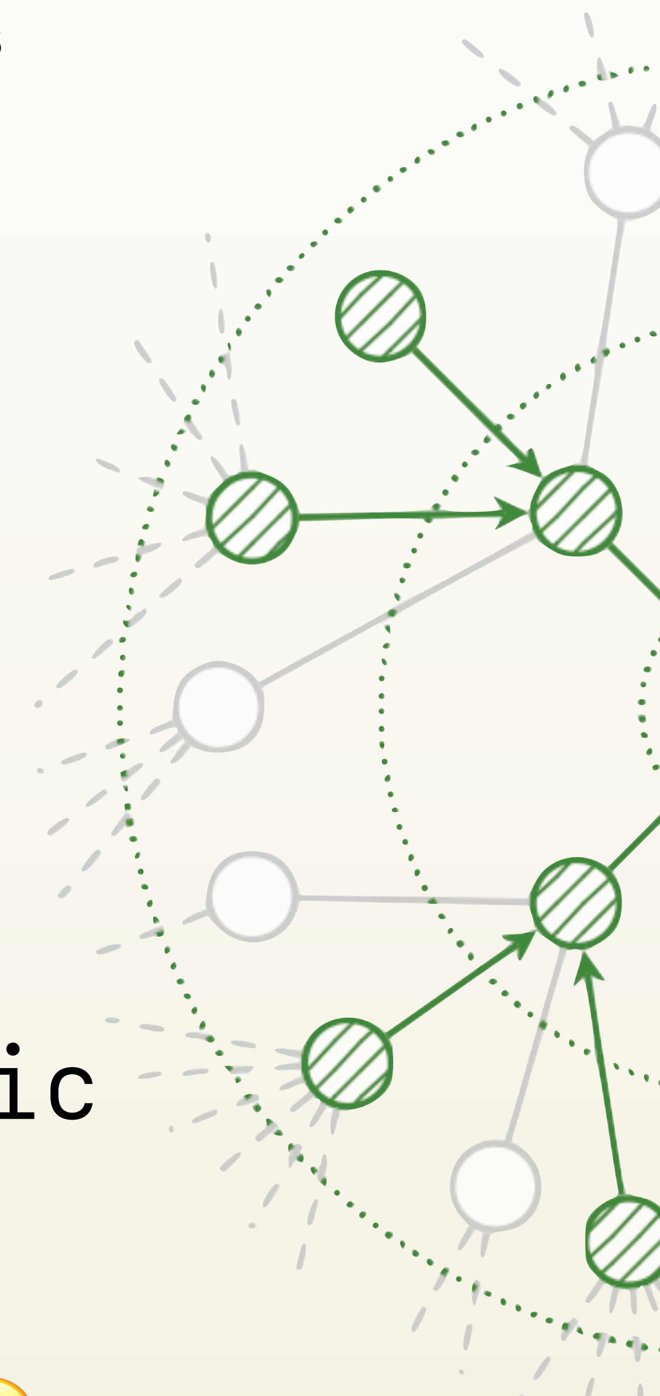
Runtimes of training procedures
for **200 epochs** on a **single GPU**

- ✓ trains most models on simple benchmark datasets in **under one second**

Dataset	Method	PyG
Cora	GCN GAT	0.25s 0.80s
CiteSeer	GCN GAT	0.30s 0.88s
PubMed	GCN GAT	0.32s 2.42s
MUTAG	RGCN	2.14s

Conclusion

- ✓ uniform implementations of over **25 GNN operators/models**
- ✓ extendable by using a simple **message passing interface**
- ✓ access to over **100 benchmark datasets**
- ✓ **dynamic** batch-wise **graph generation**
- ✓ deterministic and differentiable **pooling operators**
- ✓ basic and more sophisticated **readout functions**
- ✓ **automatic mini-batching for graphs with different sizes**
- ✓ useful transforms for **augmentation, point sampling, ...**
- ✓ leverages **dedicated CUDA kernels**
- ✓ supports **multi-GPU** setups



`/rusty1s/pytorch_geometric`

license MIT   PRs welcome

New features to come. Stay tuned! 😎