dortmund university

Stanford University

PyG 2.0

Advanced Representation Learning on Graphs

Matthias Fey

matthias@pyg.org

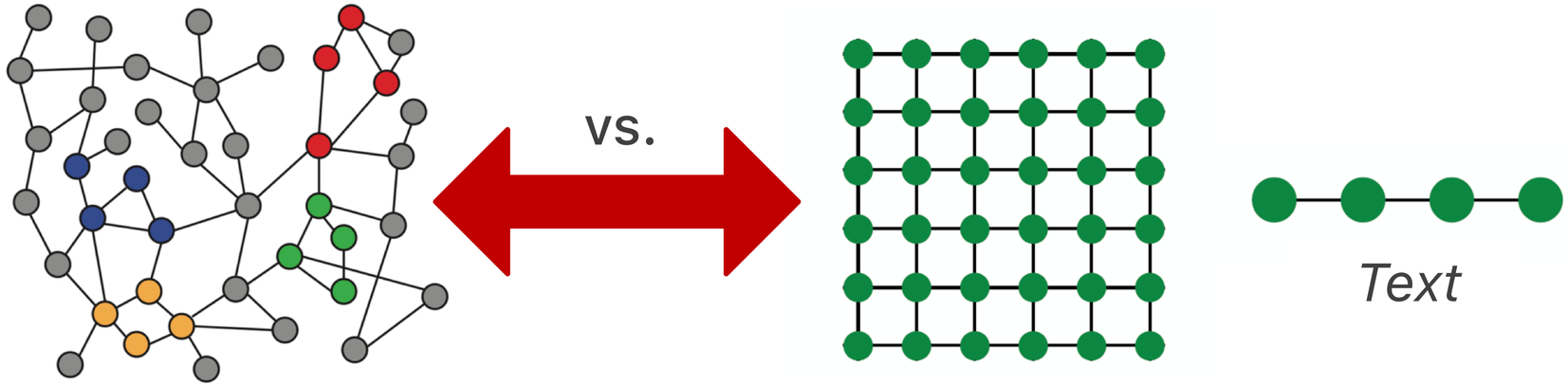
https://pyg.org

/pyg-team/pytorch-geometric


```
conda install pyg -c pyg
```

Motivation

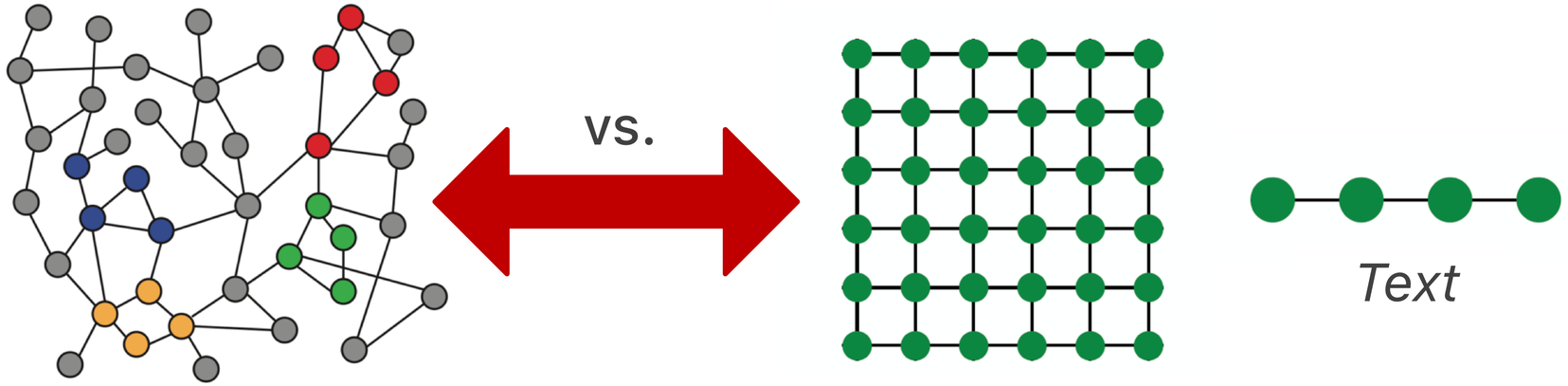
The **modern deep learning toolbox** is designed for **sequences and grids**



Graph *Image*

Motivation

The **modern deep learning toolbox** is designed for **sequences and grids**

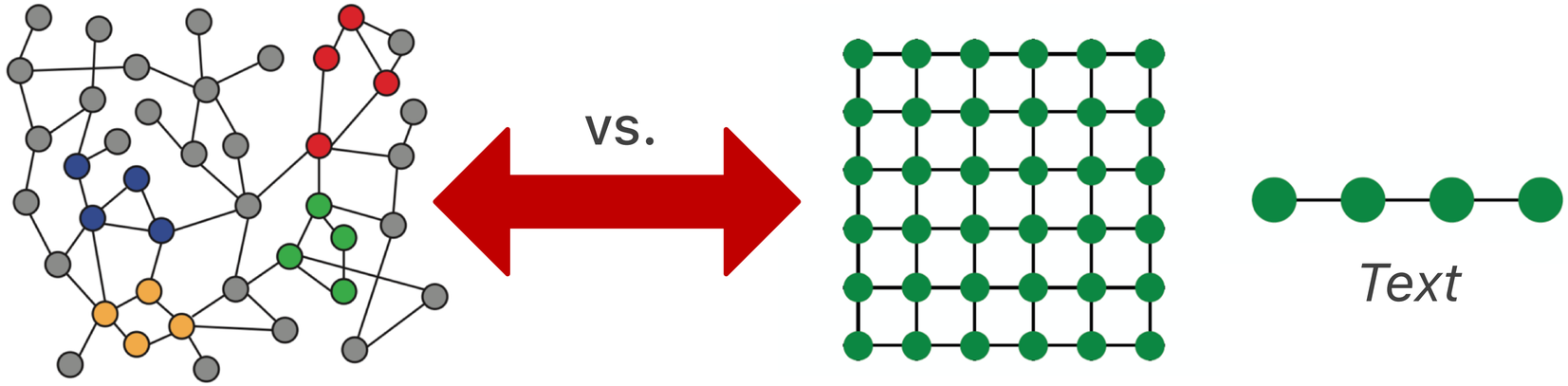


Graph *Image*

- **Arbitrary size** and **complex topological structure**

Motivation

The **modern deep learning toolbox** is designed for **sequences and grids**

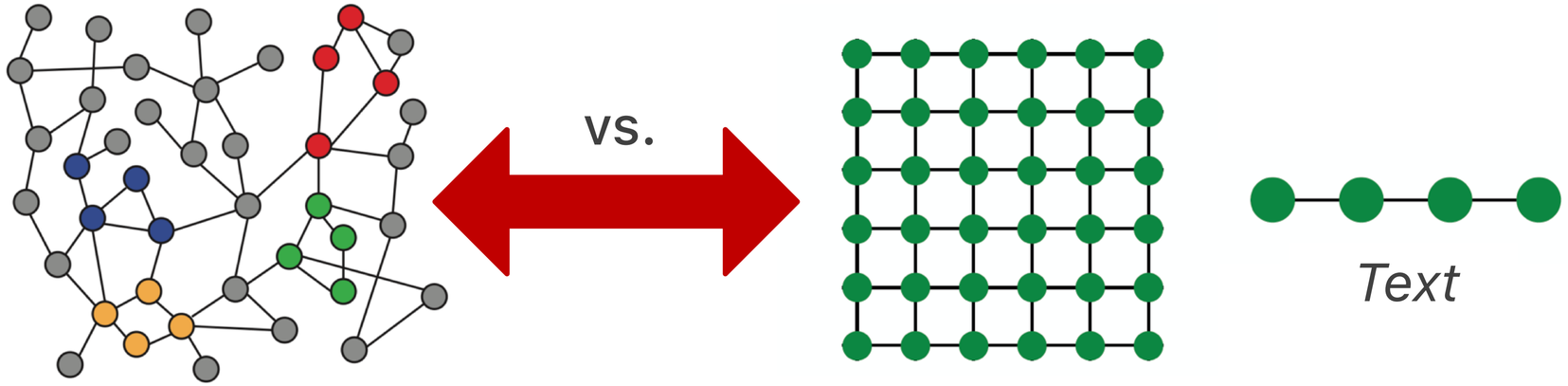


Graph *Image*

- **Arbitrary size** and **complex topological structure**
- **No fixed node ordering** or **reference point**

Motivation

The **modern deep learning toolbox** is designed for **sequences and grids**

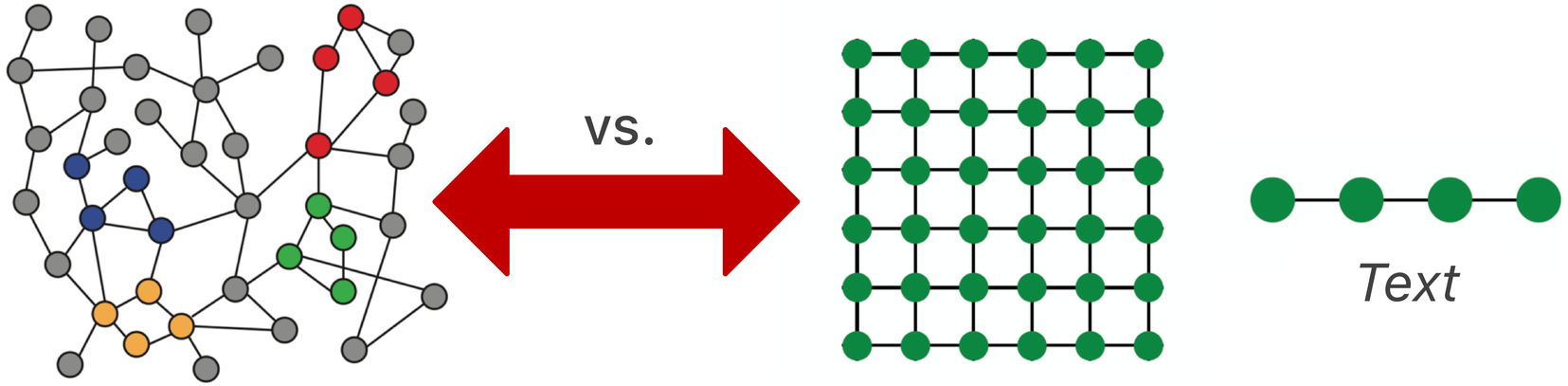


Graph *Image*

- **Arbitrary size** and **complex topological structure**
- **No fixed node ordering** or **reference point**
- **Often dynamic**

Motivation

The **modern deep learning toolbox** is designed for **sequences and grids**

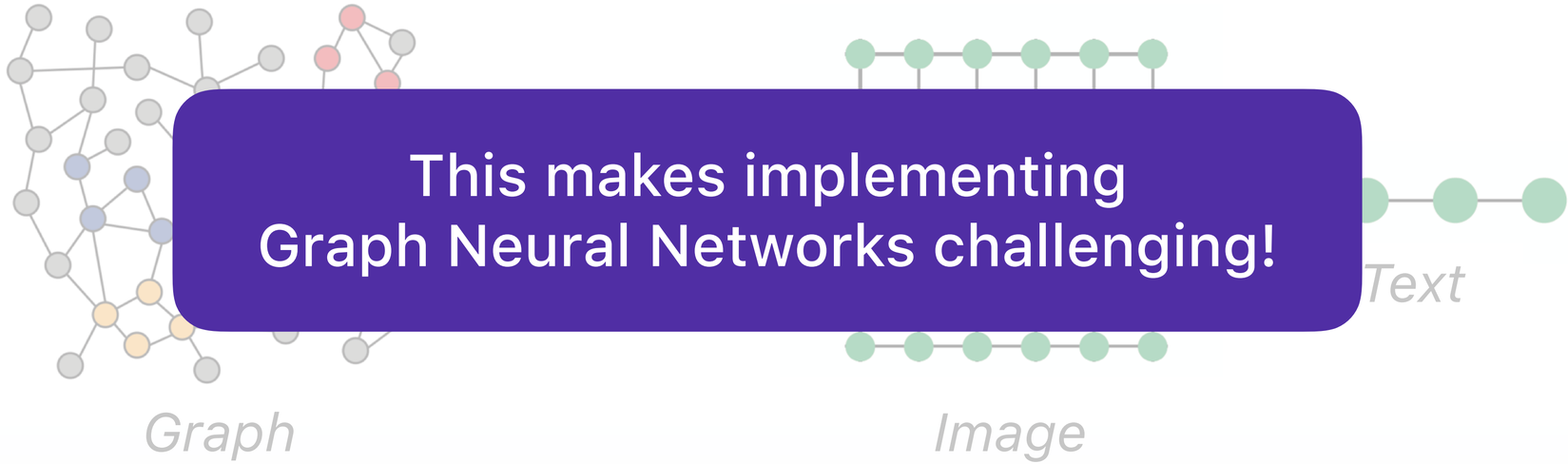


Graph *Image*

- **Arbitrary size** and **complex topological structure**
- **No fixed node ordering** or **reference point**
- **Often dynamic**
- **Multimodal node** and **edge features**

Motivation

The **modern deep learning toolbox** is designed for **sequences and grids**



- Arbitrary size and complex topological structure
- No fixed node ordering or reference point
- Often dynamic
- Multimodal node and edge features

Motivation

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

Motivation

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

- simplifies implementing and working with **Graph Neural Networks (GNNs)**

Motivation

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

- simplifies implementing and working with **Graph Neural Networks (GNNs)**
- bundles **state-of-the-art GNN architectures** and **training procedures**

Motivation

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

- simplifies implementing and working with **Graph Neural Networks (GNNs)**
- bundles **state-of-the-art GNN architectures** and **training procedures**
- achieves **high GPU throughput** on highly sparse data of varying size

Motivation

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

- simplifies implementing and working with **Graph Neural Networks (GNNs)**
- bundles **state-of-the-art GNN architectures** and **training procedures**
- achieves **high GPU throughput** on highly sparse data of varying size
- suited for both **academia** and **industry**

Design Principles

 **PyG** is  PyTorch-on-the-rocks:

Design Principles

PyG is PyTorch-on-the-rocks:

- **PyG** is framework-specific
  allows us to make use of recently released features right away:
  `TorchScript` for deployment, `torch.fx` for model transformation

Design Principles

PyG is PyTorch-on-the-rocks:

- **PyG** is framework-specific
  allows us to make use of recently released features right away:
  `TorchScript` for deployment, `torch.fx` for model transformation
- **PyG** keeps **design principles** close to vanilla PyTorch
  *If you are familiar with PyTorch, you already know most about **PyG***

Design Principles

PyG is PyTorch-on-the-rocks:

- **PyG** is **framework-specific**
  allows us to make use of recently released features right away:
  `TorchScript` for deployment, `torch.fx` for model transformation
- **PyG** keeps **design principles** close to vanilla PyTorch
  *If you are familiar with PyTorch, you already know most about **PyG***
- **PyG** fits nicely into the **PyTorch ecosystem**
  *Scaling up models via **PyTorch Lightning***
  *Explaining models via **Captum***

Design Principles

PyG is PyTorch-on-the-rocks:

```python
from torch.nn import Conv2d

class CNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = Conv2d(3, 64)
        self.conv2 = Conv2d(64, 64)

    def forward(self, input):
        h = self.conv1(input)
        h = h.relu()
        h = self.conv2(h)
        return h
```

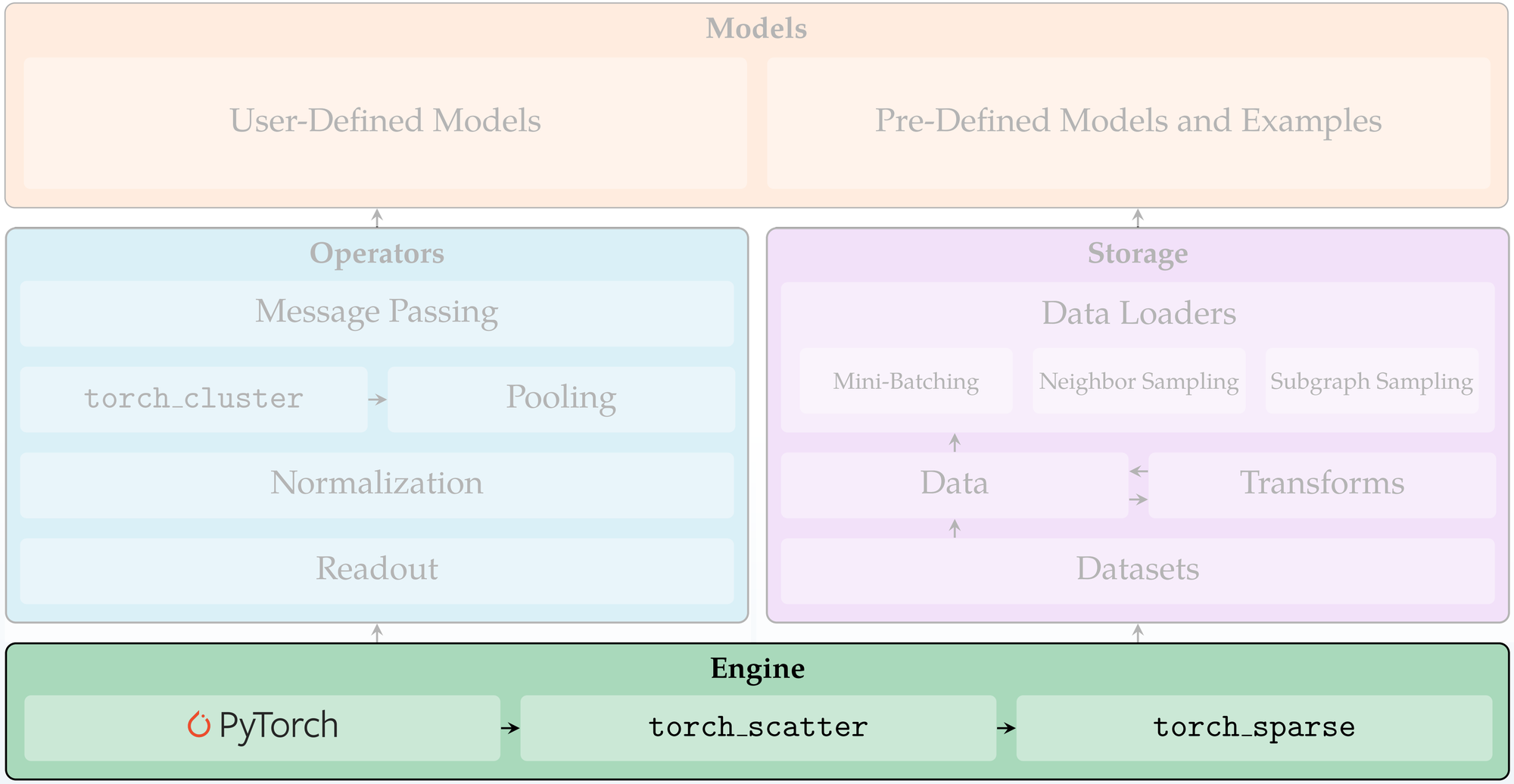
Design Principles

PyG is PyTorch-on-the-rocks:

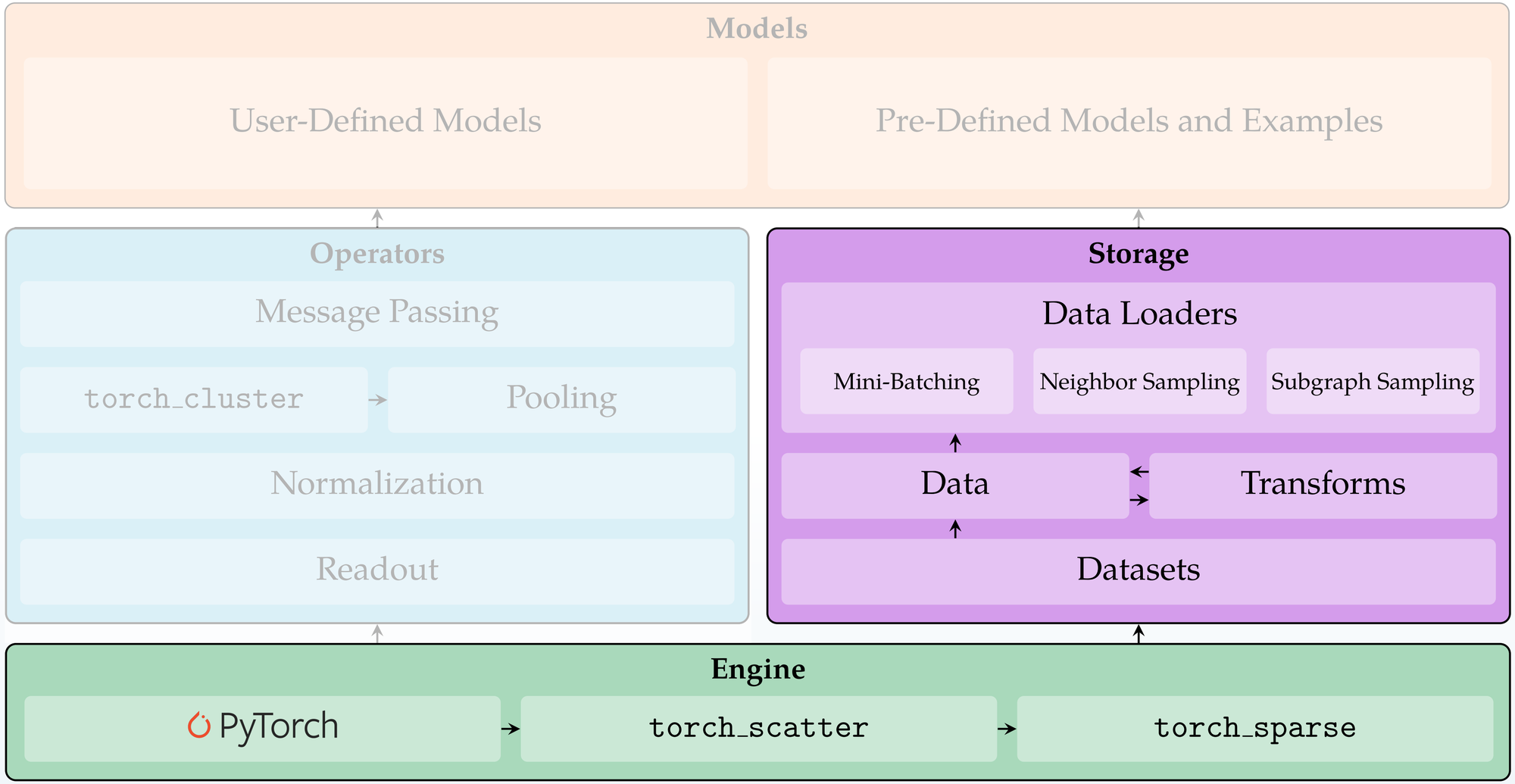
```python
from torch.nn import Conv2d

class CNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = Conv2d(3, 64)
        self.conv2 = Conv2d(64, 64)

    def forward(self, input):
        h = self.conv1(input)
        h = h.relu()
        h = self.conv2(h)
        return h
```

```python
from torch_geometric.nn import GCNConv

class GNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = GCNConv(3, 64)
        self.conv2 = GCNConv(64, 64)

    def forward(self, input, edge_index):
        h = self.conv1(input, edge_index)
        h = h.relu()
        h = self.conv2(h, edge_index)
        return h
```

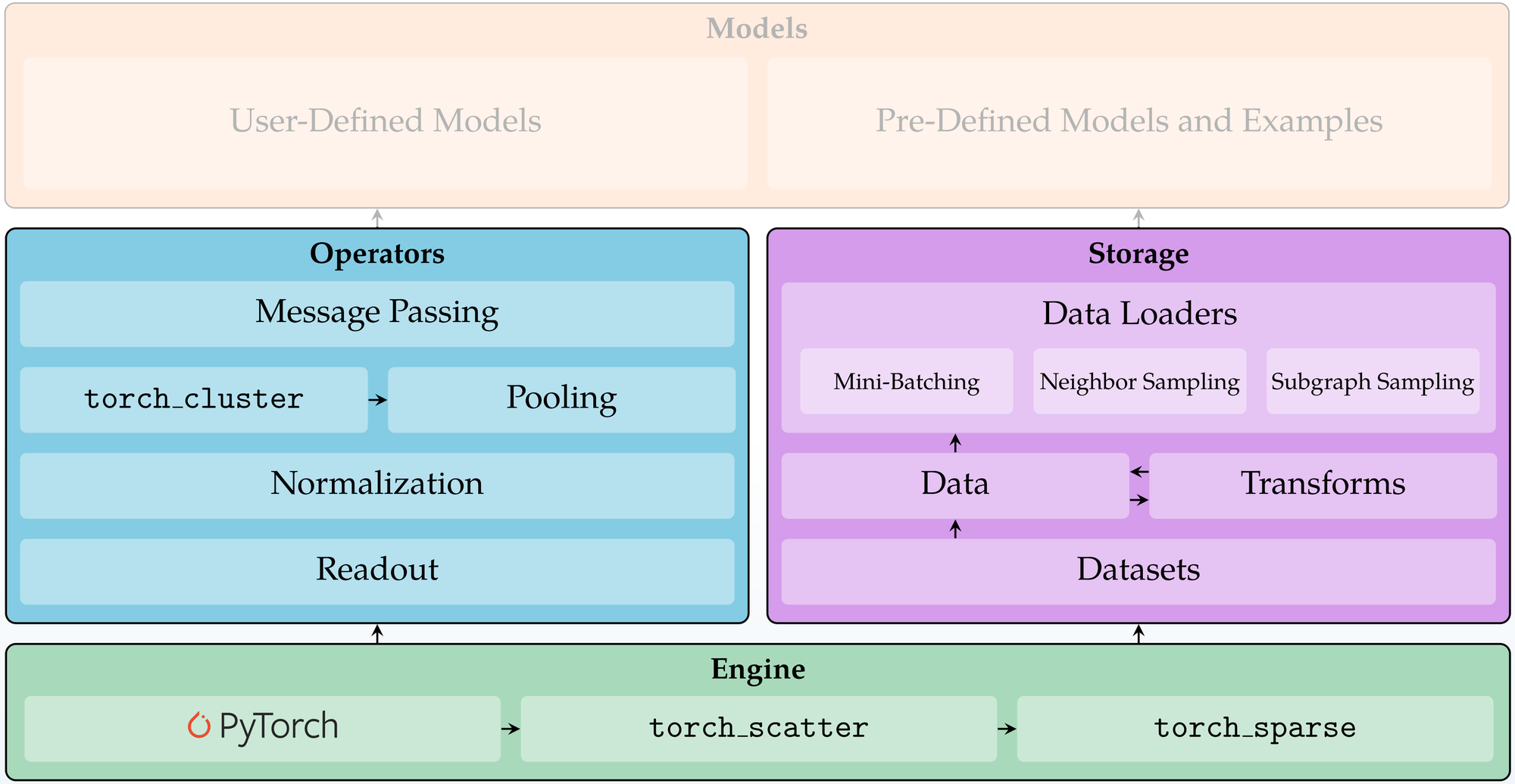
Design Principles



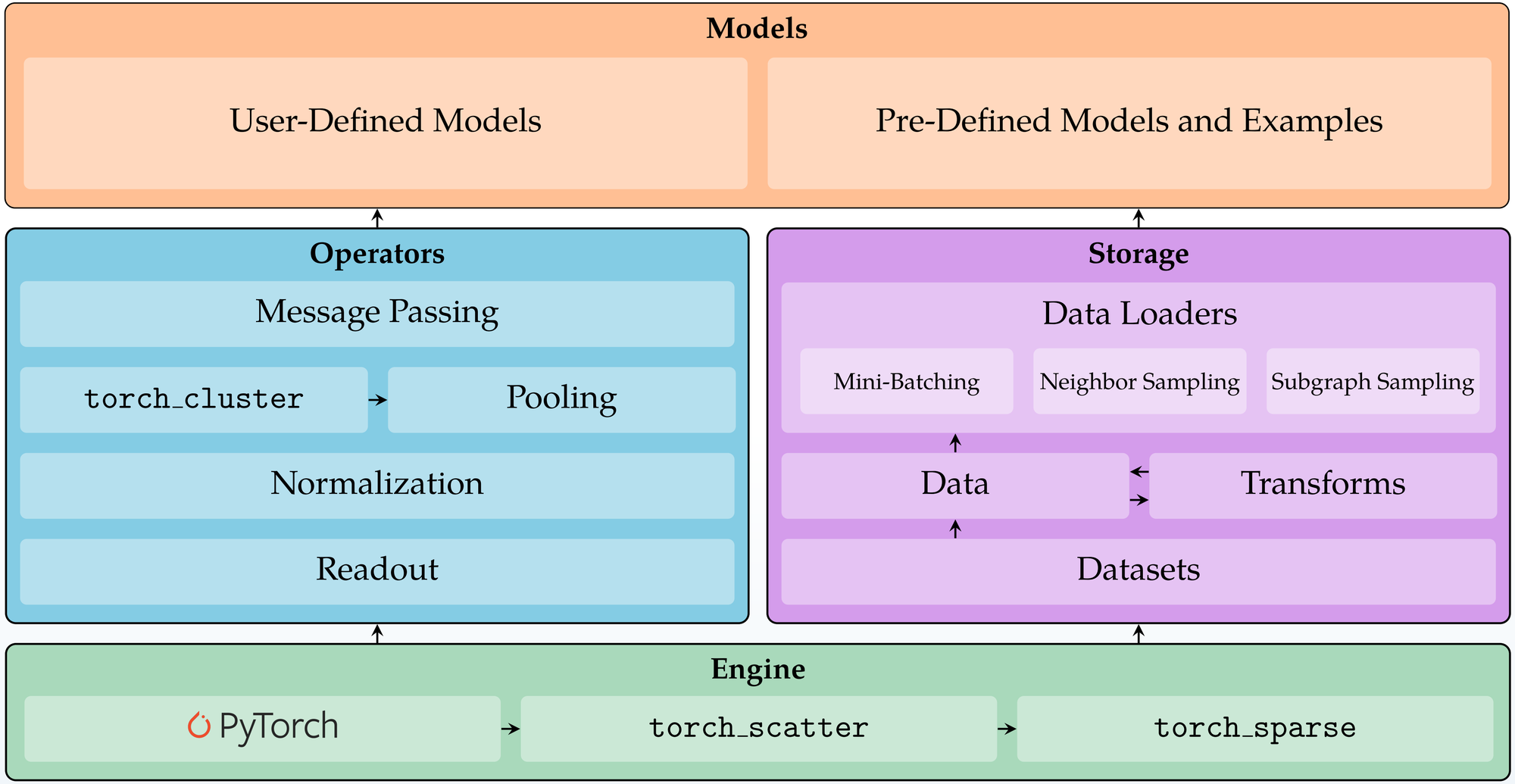
Design Principles



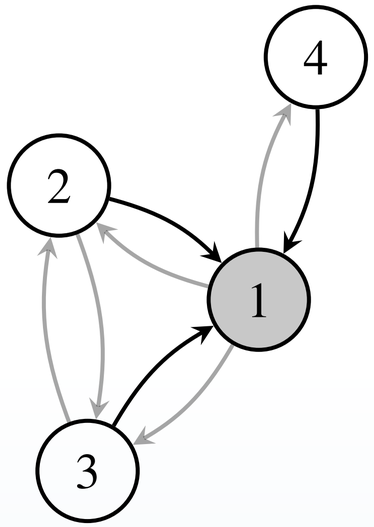
Design Principles



Design Principles



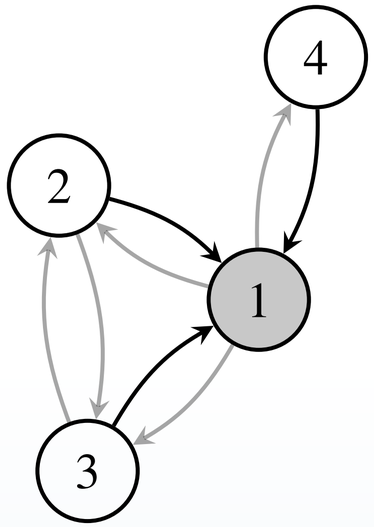
Message Passing Graph Neural Networks



Given a *sparse* graph $\mathcal{G} = (\mathbf{H}^{(0)}, (\mathbf{I}, \mathbf{E}))$ with

- input node features $\mathbf{H}^{(0)} \in \mathbb{R}^{|\mathcal{V}| \times C}$
- edge indices $\mathbf{I} \in \{1, ..., |\mathcal{V}|\}^{2 \times |\mathcal{E}|}$
- *optional* edge features $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times D}$

Message Passing Graph Neural Networks



Given a *sparse* graph $\mathcal{G} = (\mathbf{H}^{(0)}, (\mathbf{I}, \mathbf{E}))$ with

- input node features $\mathbf{H}^{(0)} \in \mathbb{R}^{|\mathcal{V}| \times C}$
- edge indices $\mathbf{I} \in \{1, ..., |\mathcal{V}|\}^{2 \times |\mathcal{E}|}$
- *optional* edge features $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times D}$

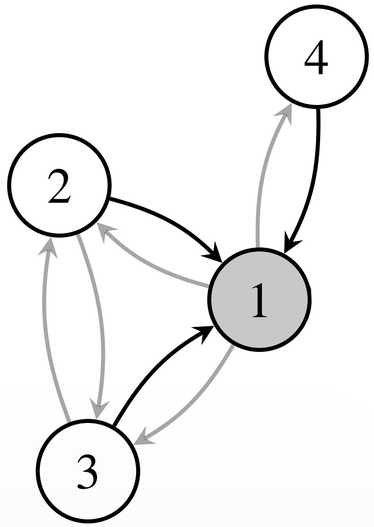
Message Passing Scheme

permutation-invariant aggregation operator, e.g., *sum*, *mean* or *max*

$$\mathbf{h}_i^{(\ell+1)} = \textsc{Update}_\theta \left(\mathbf{h}_i^{(\ell)}, \bigoplus_{j \in \mathcal{N}(i)} \textsc{Message}_\theta \left(\mathbf{h}_j^{(\ell)}, \mathbf{h}_i^{(\ell)}, \mathbf{e}_{j,i} \right) \right)$$

Message Passing Graph Neural Networks

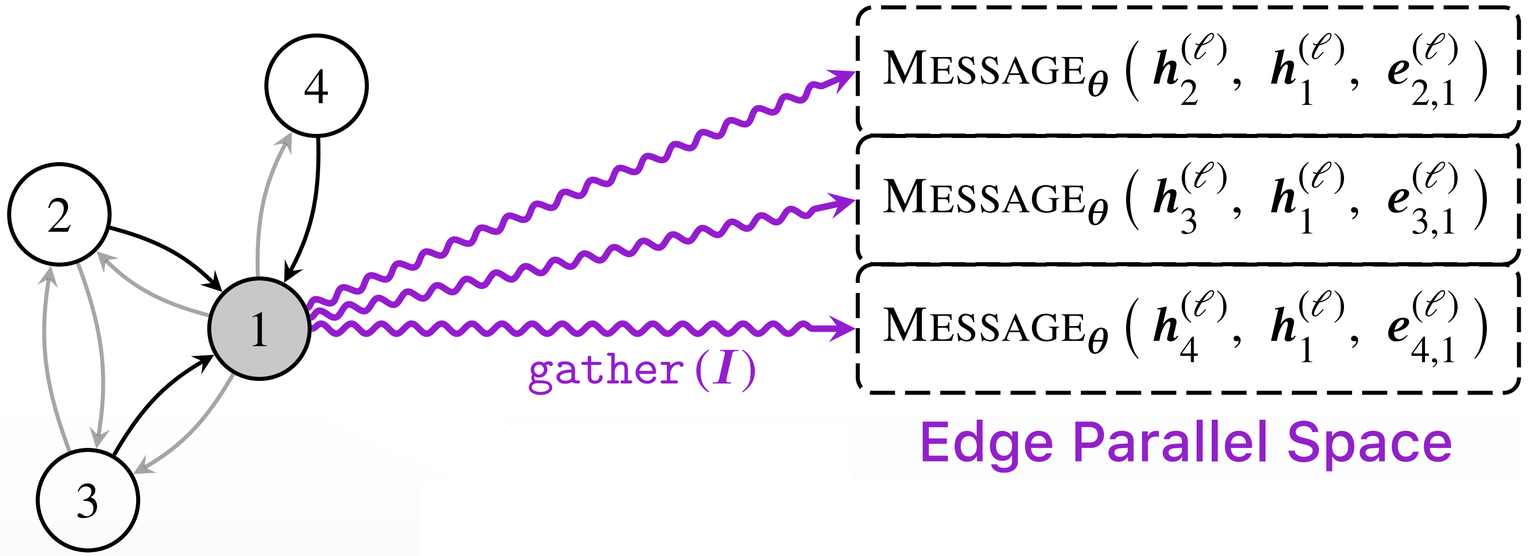
Flexible implementation via *parallelizable* `gather` and `scatter` operations condensed in a general `MessagePassing` interface



$$\mathbf{h}_i^{(\ell+1)} = \textsc{Update}_\theta \left(\mathbf{h}_i^{(\ell)}, \bigoplus_{j \in \mathcal{N}(i)} \textsc{Message}_\theta \left(\mathbf{h}_j^{(\ell)}, \mathbf{h}_i^{(\ell)}, \mathbf{e}_{j,i} \right) \right)$$

Message Passing Graph Neural Networks

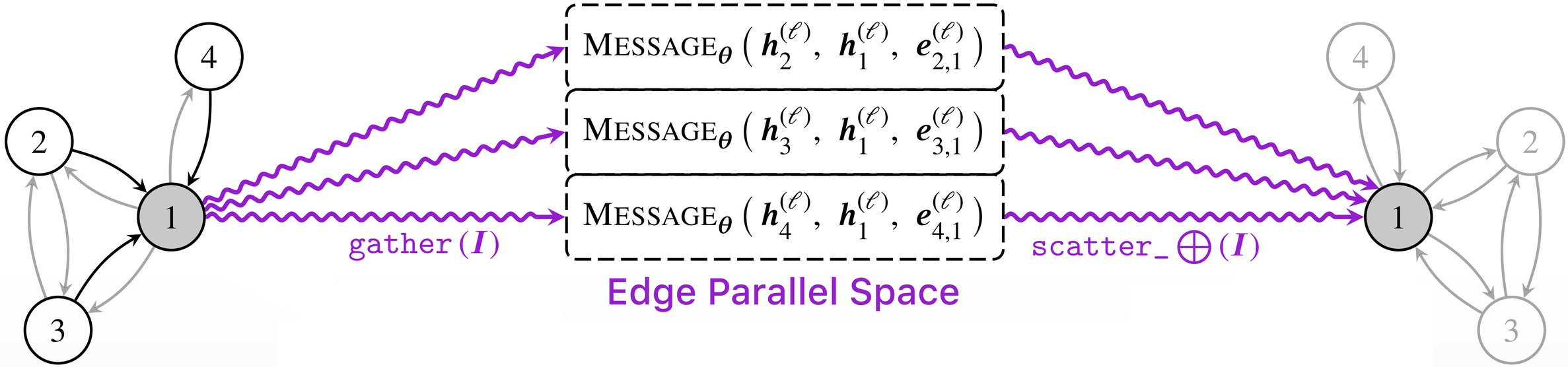
Flexible implementation via *parallelizable* `gather` and `scatter` operations condensed in a general `MessagePassing` interface



$$\mathbf{h}_i^{(\ell+1)} = \text{UPDATE}_\theta\left(\mathbf{h}_i^{(\ell)}, \bigoplus_{j \in \mathcal{N}(i)} \text{MESSAGE}_\theta\left(\mathbf{h}_j^{(\ell)}, \mathbf{h}_i^{(\ell)}, \mathbf{e}_{j,i}\right)\right)$$

Message Passing Graph Neural Networks

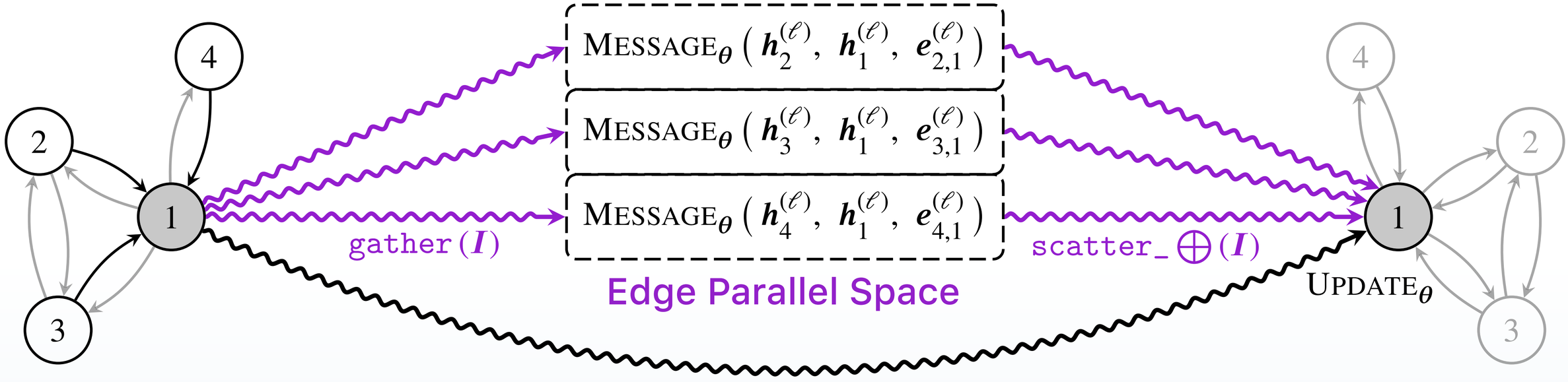
Flexible implementation via *parallelizable* `gather` and `scatter` operations condensed in a general `MessagePassing` interface



$$\mathbf{h}_i^{(\ell+1)} = \text{UPDATE}_\theta\left(\mathbf{h}_i^{(\ell)}, \bigoplus_{j \in \mathcal{N}(i)} \text{MESSAGE}_\theta\left(\mathbf{h}_j^{(\ell)}, \mathbf{h}_i^{(\ell)}, \mathbf{e}_{j,i}\right)\right)$$

Message Passing Graph Neural Networks

Flexible implementation via *parallelizable* `gather` and `scatter` operations condensed in a general `MessagePassing` interface



$$\mathbf{h}_i^{(\ell+1)} = \textsc{Update}_\theta\left(\mathbf{h}_i^{(\ell)}, \bigoplus_{j \in \mathcal{N}(i)} \textsc{Message}_\theta\left(\mathbf{h}_j^{(\ell)}, \mathbf{h}_i^{(\ell)}, \mathbf{e}_{j,i}\right)\right)$$

Message Passing Graph Neural Networks

PyG covers a **large number of state-of-the-art GNN layers and architectures**, and can easily be extended to fit to a specific use-case

Message Passing Graph Neural Networks

PyG covers a **large number of state-of-the-art GNN layers and architectures**, and can easily be extended to fit to a specific use-case

GNN layers:

Cheby GCN SAGE PointNet MoNet MPNN GAT SuperGAT
SplineCNN AGNN EdgeCNN S-GCN R-GCN PointCNN
ARMA APPNP GIN GIN-E CG GatedGCN NMF TAG DNA
Signed-GCN PPFNet FeaST Hyper-GCN GravNet PDN WL
ResGatedGCN SparseTransformer DGCNN FeaST EG LeCNN
PNA GEN GCN2 PAN FiLM SuperGAT FA HGT

Message Passing Graph Neural Networks

PyG covers a **large number of state-of-the-art GNN layers and architectures**, and can easily be extended to fit to a specific use-case

GNN layers:

Cheby GCN SAGE PointNet MoNet MPNN GAT SuperGAT
SplineCNN AGNN EdgeCNN S-GCN R-GCN PointCNN
ARMA APPNP GIN GIN-E CG GatedGCN NMF TAG DNA
Signed-GCN PPFNet FeaST Hyper-GCN GravNet PDN WL
ResGatedGCN SparseTransformer DGCNN FeaST EG LeCNN
PNA GEN GCN2 PAN FiLM SuperGAT FA HGT

GNN models:

GAE ARGA DGI Node2Vec Graph-UNet AFP C&S LP RECT
GeniePath SchNet DimeNet MetaPath2Vec RENet TGN SEAL

Message Passing Graph Neural Networks

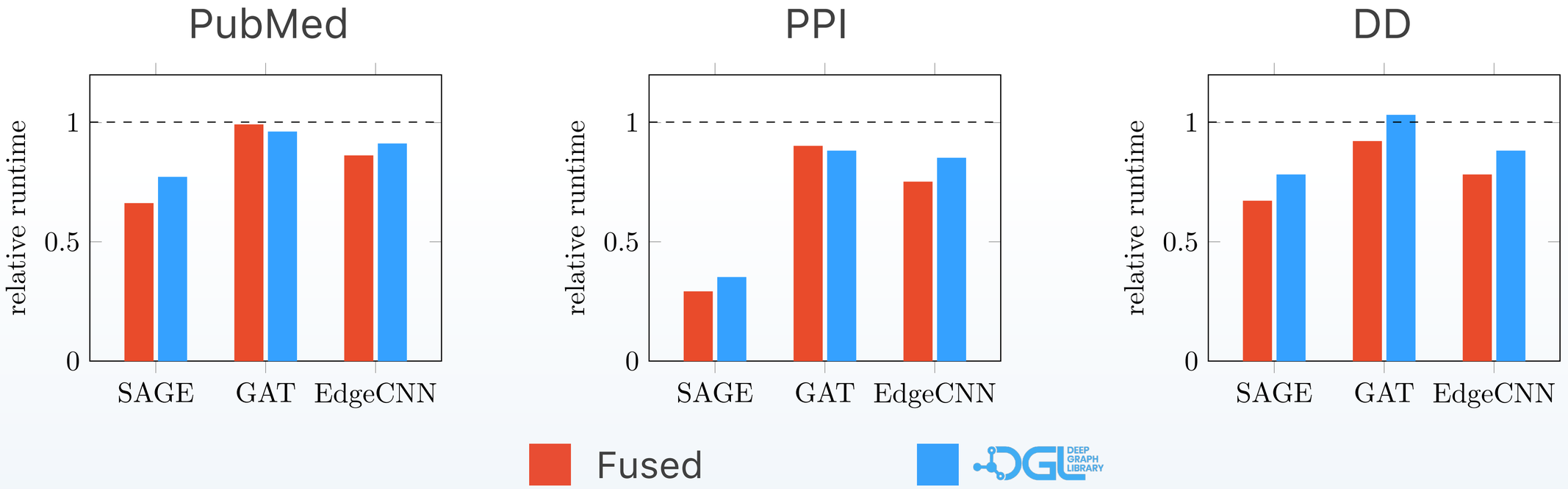
Not all GNN operators need to **explicitly materialize** the edge parallel space

In this case, **PyG** can **fuse message and aggregation computation** for better memory efficiency: $\mathcal{O}(|\mathcal{V}|)$

Message Passing Graph Neural Networks

Not all GNN operators need to **explicitly materialize** the edge parallel space

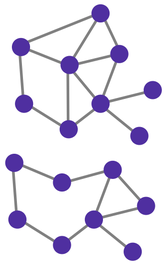
In this case, **PyG** can **fuse message and aggregation computation** for better memory efficiency: $\mathcal{O}(|\mathcal{V}|)$



Mini-Batching in GNNs

PyG supports mini-batching on **many small graphs**

$$\mathcal{G}_1 = (\mathbf{H}_1, \mathbf{A}_1)$$
$$\mathcal{G}_2 = (\mathbf{H}_2, \mathbf{A}_2)$$

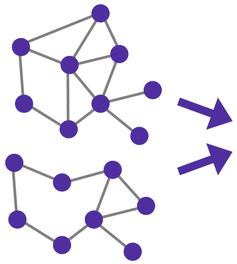


$$\mathrm{GNN}\left(\begin{bmatrix} \mathbf{A}_1 & \\ & \mathbf{A}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_2 \end{bmatrix}\right) = \begin{bmatrix} \mathbf{H}'_1 \\ \mathbf{H}'_2 \end{bmatrix}$$

Mini-Batching in GNNs

PyG supports mini-batching on **many small graphs**

$$\mathcal{G}_1 = (\mathbf{H}_1, \mathbf{A}_1)$$
$$\mathcal{G}_2 = (\mathbf{H}_2, \mathbf{A}_2)$$



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

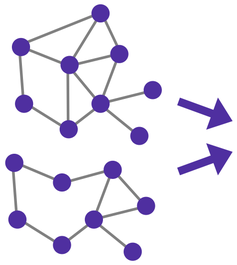
- ✓ no GNN **modifications** needed
- ✓ **no memory/computation overhead**
- ✓ supports examples of **varying size**

Mini-Batching in GNNs

PyG supports mini-batching on **many small graphs**

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

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



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

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

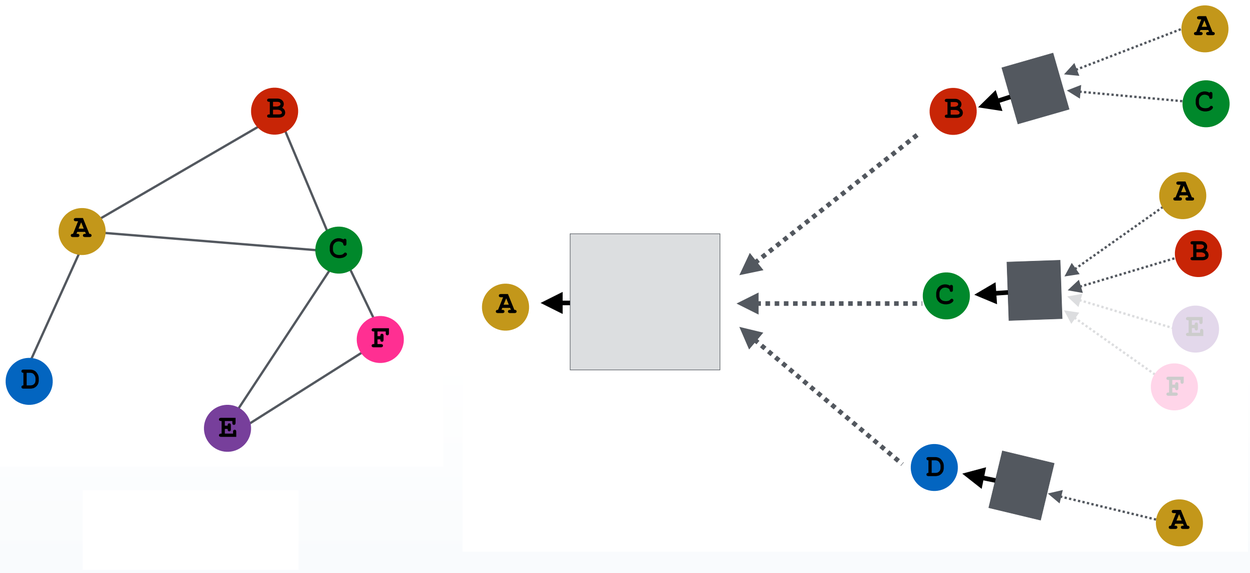
```python
from torch_geometric.datasets import TUDataset
from torch_geometric.loader import DataLoader

dataset = TUDataset(name='IMDB-BINARY')
loader = DataLoader(dataset, batch_size=128)
```

Mini-Batching in GNNs

PyG supports mini-batching on **single giant graphs**

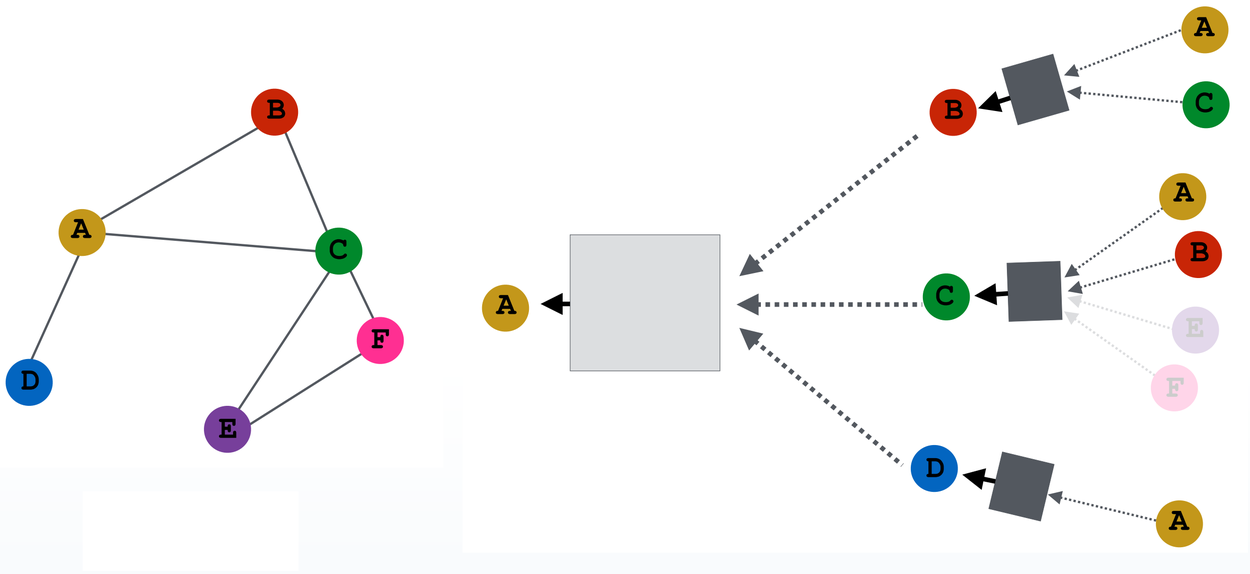
Mini-Batching in GNNs

PyG supports mini-batching on **single giant graphs**



Mini-Batching in GNNs

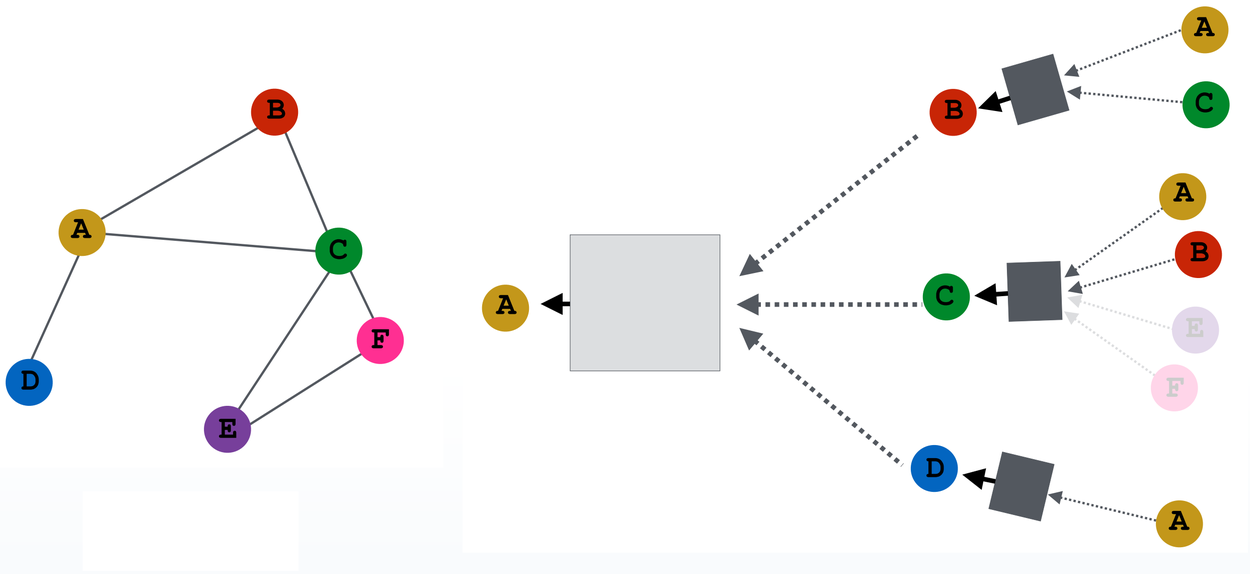
PyG supports mini-batching on **single giant graphs**




```python
from torch_geometric.datasets import Reddit
from torch_geometric.loader import NeighborLoader

data = Reddit('data/Reddit')[0]
loader = NeighborLoader(data, batch_size=128,
                        num_neighbors=[25, 10])

for batch in loader:
    model(batch.inputs, batch.edge_index)
```

Mini-Batching in GNNs

PyG supports mini-batching on **single giant graphs**




```python
from torch_geometric.datasets import Reddit
from torch_geometric.loader import NeighborLoader

data = Reddit('data/Reddit')[0]
loader = NeighborLoader(data, batch_size=128,
                        num_neighbors=[25, 10])

for batch in loader:
    model(batch.inputs, batch.edge_index)
```

Scalability support:

NeighborSampling ClusterGCN GraphSAINT ShaDow
SGC SIGN Correct&Smooth GNNAutoScale

Graph Machine Learning Toolkit

 **PyG** can handle **standard graph learning tasks** with ease ...

Graph Machine Learning Toolkit

PyG can handle **standard graph learning tasks** with ease ...

Node-level Predictions

Predict the class of a node

$$\phi(\mathcal{G}, v) \in [0, 1]^C$$

Graph Machine Learning Toolkit

PyG can handle **standard graph learning tasks** with ease ...

Node-level Predictions

Predict the class of a node

$$\phi(\mathcal{G}, v) \in [0, 1]^C$$

Link-level Predictions

Predict the class of a link

$$\phi(\mathcal{G}, v, w) \in [0, 1]^C$$

Graph Machine Learning Toolkit

PyG can handle **standard graph learning tasks** with ease ...

Node-level Predictions

Predict the class of a node

$$\phi(\mathcal{G}, v) \in [0, 1]^C$$

Link-level Predictions

Predict the class of a link

$$\phi(\mathcal{G}, v, w) \in [0, 1]^C$$

Graph-level Predictions

Predict the class of a graph

$$\phi(\mathcal{G}) \in [0, 1]^C$$

Graph Machine Learning Toolkit

PyG can handle **standard graph learning tasks** with ease ...

Node-level Predictions

Predict the class of a node

$$\phi(\mathcal{G}, v) \in [0, 1]^C$$

Link-level Predictions

Predict the class of a link

$$\phi(\mathcal{G}, v, w) \in [0, 1]^C$$

Graph-level Predictions

Predict the class of a graph

$$\phi(\mathcal{G}) \in [0, 1]^C$$

... but is **not limited** to those:

Graph Machine Learning Toolkit

PyG can handle **standard graph learning tasks** with ease ...

Node-level Predictions
Predict the class of a node

$$\phi(\mathcal{G}, v) \in [0, 1]^C$$

Link-level Predictions
Predict the class of a link

$$\phi(\mathcal{G}, v, w) \in [0, 1]^C$$

Graph-level Predictions
Predict the class of a graph

$$\phi(\mathcal{G}) \in [0, 1]^C$$

... but is **not limited** to those:

Unsupervised Learning Self-Supervised Learning
Few/Zero-Shot Learning Pre-Training Explainability ...

Graph Machine Learning Toolkit

PyG can handle **standard graph learning tasks** with ease ...

Node-level Predictions
Predict the class of a node
$\phi(\mathcal{G}, v) \in [0, 1]^C$

Link-level Predictions
Predict the class of a link
$\phi(\mathcal{G}, v, w) \in [0, 1]^C$

Graph-level Predictions
Predict the class of a graph
$\phi(\mathcal{G}) \in [0, 1]^C$

... but is **not limited** to those:

Unsupervised Learning **Self-Supervised Learning**
Few/Zero-Shot Learning **Pre-Training** **Explainability** ...

PyG provides over **80 examples** with access to over **200 benchmark datasets** to get familiar with the **latest trends in graph machine learning**

Additional Features

Additional Features

Convert pure Python GNN model to an **optimized** and **standalone** program

```
class GNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = GCNConv( ... ).jittable()
        self.conv2 = GCNConv( ... ).jittable()

model = torch.jit.script(GNN())
```

Additional Features

TorchScript

Convert pure Python GNN model to an **optimized** and **standalone** program

```python
class GNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = GCNConv( ... ).jittable()
        self.conv2 = GCNConv( ... ).jittable()

model = torch.jit.script(GNN())
```

PyTorch Lightning

1. Choose a **GNN model**
2. Setup a **trainer**
   (#GPUs, accelerator type)
3. Call `trainer.fit()`

```python
data = Reddit('data/Reddit')
model = GraphSAGE(in_channels, out_channels)

trainer = Trainer(gpus=2, accelerator='ddp')

trainer.fit(model, data)
trainer.test()
```

Additional Features

TorchScript

Convert pure Python GNN model to an **optimized** and **standalone** program

```
class GNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = GCNConv( ... ).jittable()
        self.conv2 = GCNConv( ... ).jittable()

model = torch.jit.script(GNN())
```

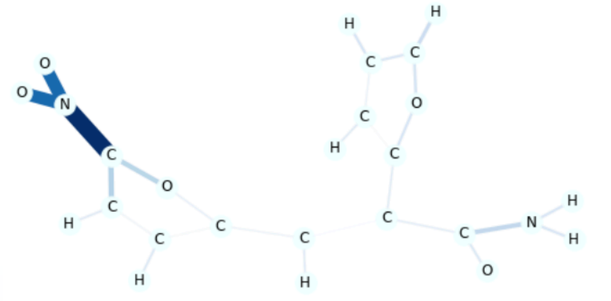
PyTorch Lightning

1. Choose a **GNN model**
2. Setup a **trainer** (#GPUs, accelerator type)
3. Call `trainer.fit()`

```
data = Reddit('data/Reddit')
model = GraphSAGE(in_channels, out_channels)

trainer = Trainer(gpus=2, accelerator='ddp')

trainer.fit(model, data)
trainer.test()
```

Captum

Explain GNN predictions out-of-the-box



```
from captum.attr import IntegratedGradients

ig = IntegratedGradients(model)
ig.attribute(input, edge_index, target=0)
```

Success Stories

- ~700 research papers using PyG
- ~2M weekly wheel downloads
- ~180 external contributors
- ~600 members on Slack

Yann LeCun
@ylecun

A fast & nice-looking PyTorch library for geometric deep learning (NN on graphs and other irregular structures).

Thomas Kipf
@thomaskipf

PyTorch Geometric has been growing into a fantastic library for graph neural nets and related methods.

Success Stories

- ~700 research papers using **PyG**
- ~2M weekly wheel downloads
- ~180 external contributors
- ~600 members on Slack

Yann LeCun
@ylecun

A fast & nice-looking PyTorch library for geometric deep learning (NN on graphs and other irregular structures).

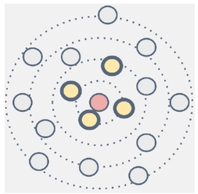
Thomas Kipf
@thomaskipf

PyTorch Geometric has been growing into a fantastic library for graph neural nets and related methods.

PyG already has its own ecosystem:

PyTorch Geometric Temporal

Dive into Graphs



DeepGCNs

Open Catalyst Project

FACEBOOK AI

Success Stories

OPEN GRAPH BENCHMARK

https://ogb.stanford.edu
/snap-stanford/ogb

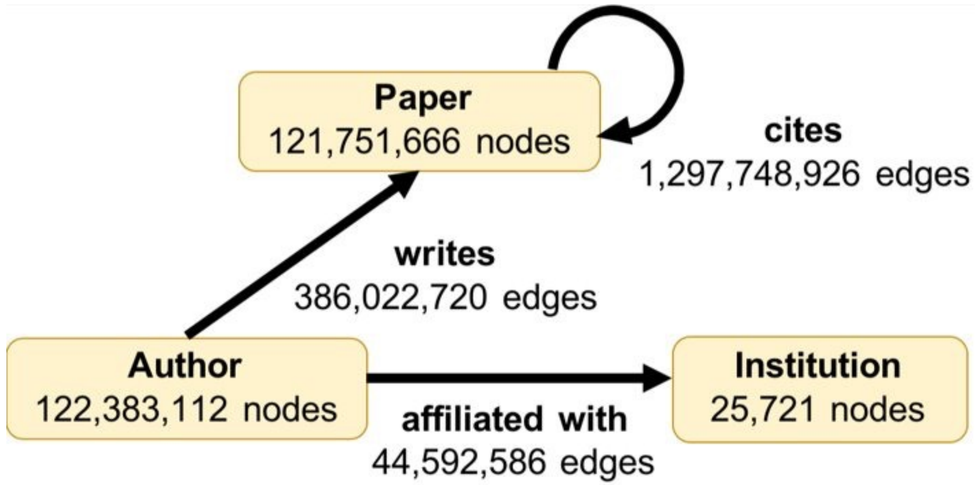
- OGB provides a variety of **realistic** and **large-scale** graph benchmark datasets
- Data loaders are **compatible** with **PyG**
- Dive-in examples are **utilizing** **PyG**

```python
from ogb import PygGraphPropPredDataset
from torch_geometric.loader import DataLoader

dataset = PygGraphPropPredDataset('ogbg-molhiv')
loader = DataLoader(dataset, batch_size=128)
```

Success Stories

OGB-LSC — Large-Scale Challenge @ KDD Cup 2021

Competition on **three large-scale graph datasets**, *e.g.*:

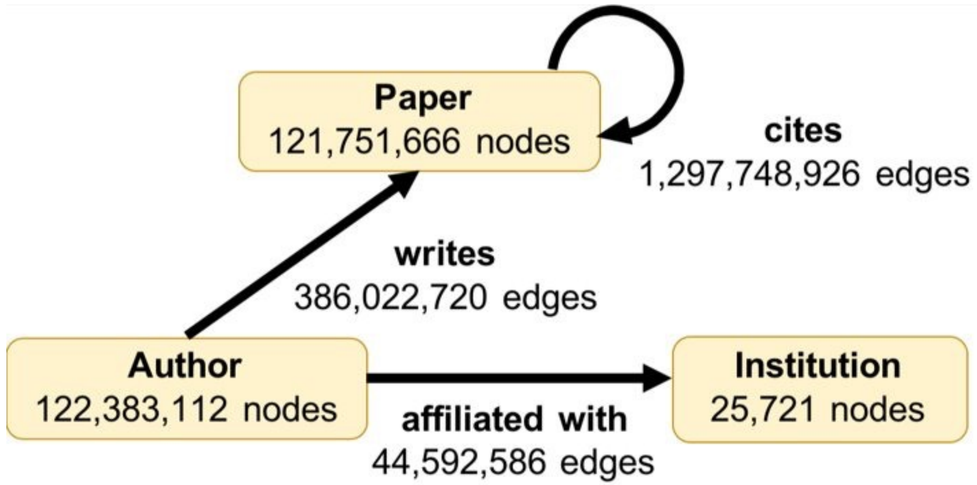


- **~240M nodes** across three node types
- **~1.7B edges** across three edge types

Success Stories

OGB-LSC — Large-Scale Challenge @ KDD Cup 2021

Competition on **three large-scale graph datasets**, *e.g.*:



- **~240M nodes** across three node types
- **~1.7B edges** across three edge types

We used PyG to benchmark GNNs:

Model	#Params	Validation	Test
MLP	0.5M	52.67	52.73
LabelProp	0	58.44	56.29
SGC	0.7M	65.82	65.29
SIGN	3.8M	66.64	66.09
MLP+C&S	0.5M	66.98	66.18
GraphSAGE (NS)	4.9M	66.79	66.28
GAT (NS)	4.9M	67.15	66.80
R-GraphSAGE (NS)	12.2M	69.86	68.94
R-GAT (NS)	12.3M	**70.02**	**69.42**

Joining Forces

We have build a **larger team** for the **future development** of **PyG**, in close collaboration with

dortmund university

Stanford University

https://pyg.org

- ensure **longevity** of **PyG**
- integrate tools from both parties into a **unified package**
- keep up with **integrating latest trends** in academic research
- extend **its scope** to better support real-world use-cases
- make it *(even)* **easier to use** for both academia and industry
- make it *(even)* **more scalable**

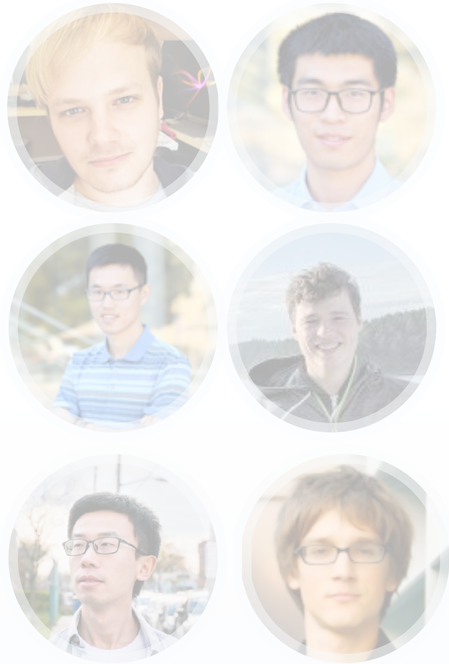
Joining Forces

We have build a **larger team** for the
future development of **PyG**,
in close collaboration with

We start with this by introducing ...

- ensure **longevity**
- integrate tools from both parties into a **unified package**
- keep up with **integrating latest trends** in academic research
- extend **its scope** to better support real-world use-cases
- make it *(even)* **easier to use** for both academia and industry
- make it *(even)* **more scalable**

https://pyg.org



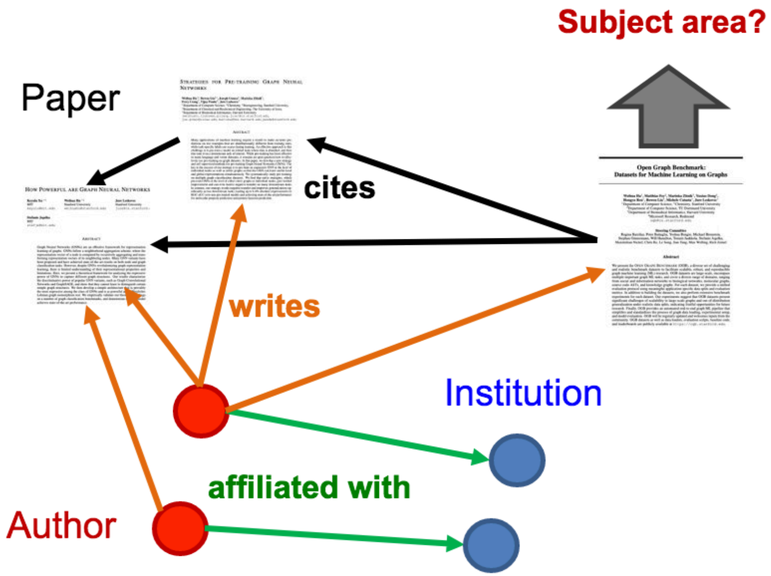
dortmund university

Stanford University

PyG 2.0

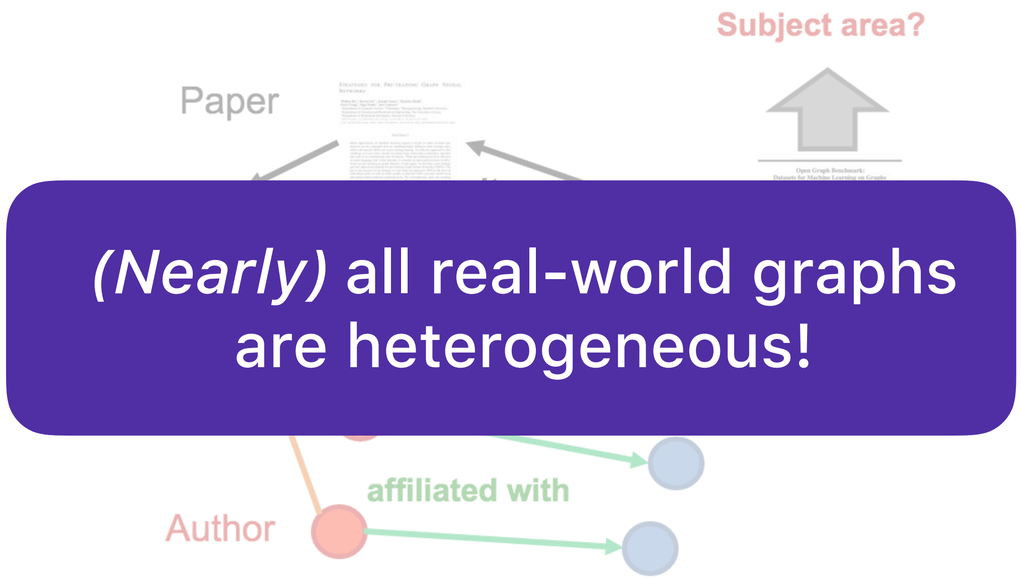
Advanced Representation Learning on Graphs

Overview



Heterogeneous Graphs

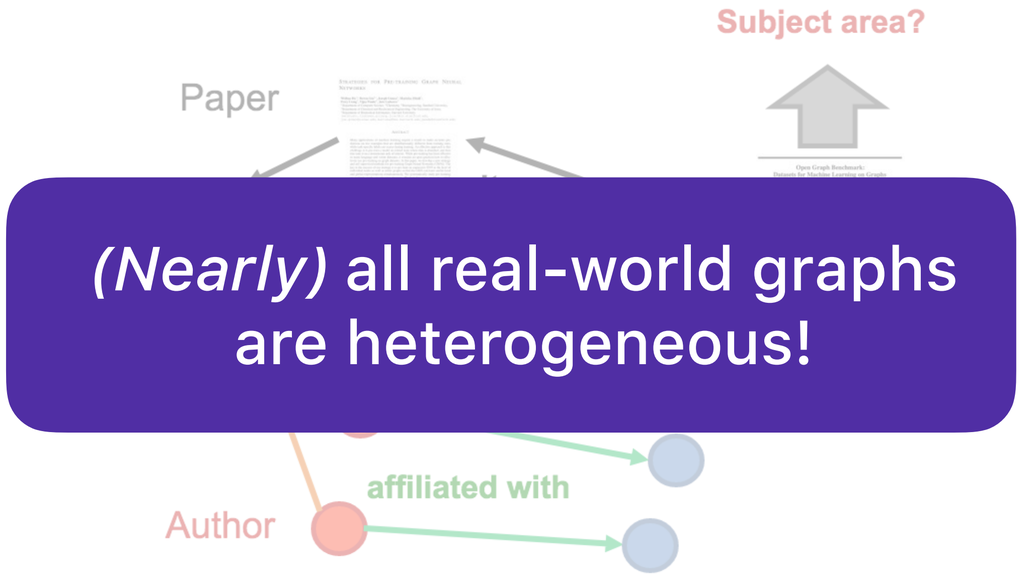
Overview



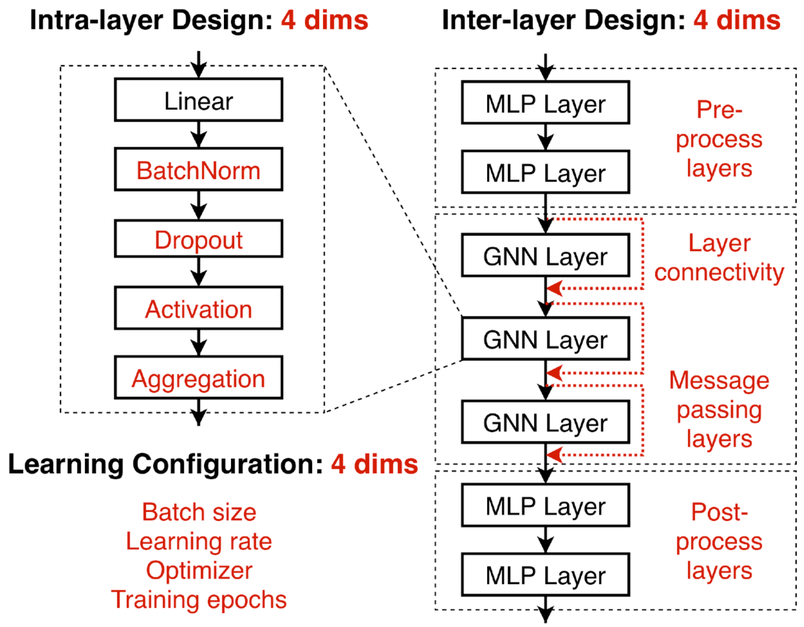
(Nearly) all real-world graphs are heterogeneous!

Heterogeneous Graphs

Overview

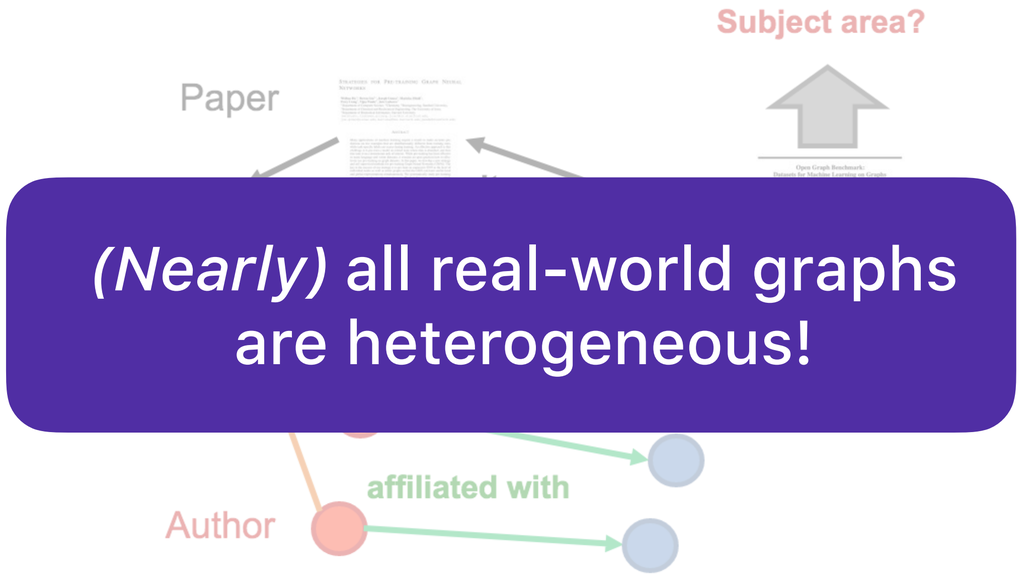


Heterogeneous Graphs

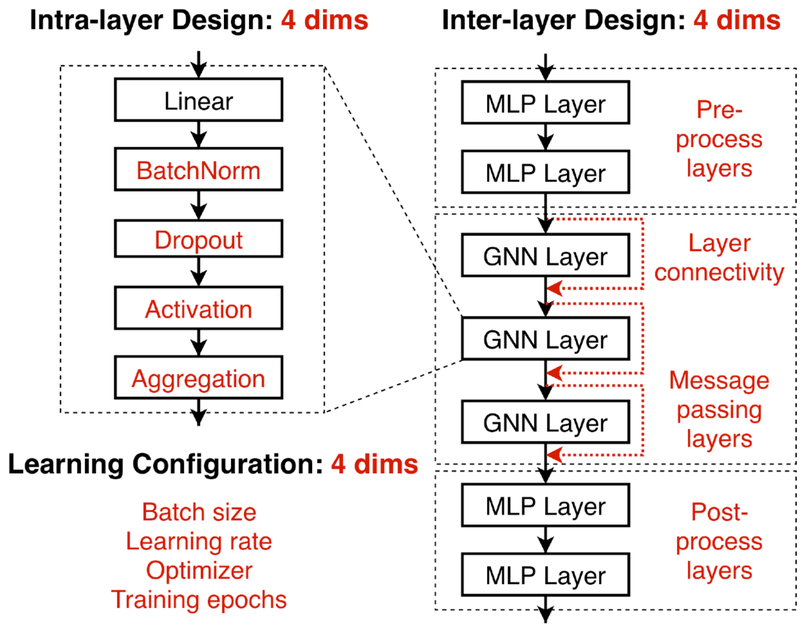


GraphGym

Overview



Heterogeneous Graphs



GraphGym

- ✓ Half-Precision Support
- ✓ *.`profile` for **benchmarking runtimes** and **memory consumptions** of GNNs
- ✓ A variety of new **operators, models, datasets** and **examples**

Heterogeneous Graph Support

Heterogeneous graph learning is notoriously challenging

Heterogeneous Graph Support

Heterogeneous graph learning is notoriously challenging

- **Different input feature distributions** across node and edge types
- **Necessity of learning node/edge type dependent representations**
  non-shared weights across different node and edge types
  bipartite message passing
- **Heterogeneous scalability approaches**
  Relational-wise neighborhood sampling
- **Complicated implementation**
  requires sequentially iterating over different node and edge types
  involves keeping track of different input feature dimensionalities

Heterogeneous Graph Support

Heterogeneous graph learning is notoriously challenging

- **Different input feature distributions** across node and edge types
- **Necessity of learning node/edge type dependent representations**
  non-shared weights across different node and edge types
  bipartite message passing
- **Heterogeneous scalability approaches**
  Relational-wise neighborhood sampling
- **Complicated implementation**
  requires sequentially iterating over different node and edge types
  involves keeping track of different input feature dimensionalities

PyG makes working with heterogeneous graphs a breeze

Heterogeneous Graph Support

Data Storage

Holds information about different node and edge types in **individual containers**

Edge types are described by a **triplet** of **source node**, **relation** and **destination node type**

Transformations enhance the graph for message passing, *e.g.*, by *adding reverse edges*

```python
from torch_geometric.data import HeteroData

data = HeteroData()

data['user'].x = ...  # User node feature matrix
data['product'].x = ...  # Product node feature matrix

# Connecting user and product nodes via "buys" relation:
data['user', 'buys', 'product'].edge_index = ...  # [2, num_edges]

# Adding reverse edges:
from torch_geometric.transforms import ToUndirected

data = ToUndirected()(data)
```

Heterogeneous Graph Support

Data Storage

Holds information about different node and edge types in **individual containers**

Edge types are described by a **triplet** of **source node**, **relation** and **destination node type**

Transformations enhance the graph for message passing, *e.g.*, by *adding reverse edges*

```python
from torch_geometric.data import HeteroData

data = HeteroData()

data['user'].x = ...  # User node feature matrix
data['product'].x = ...  # Product node feature matrix

# Connecting user and product nodes via "buys" relation:
data['user', 'buys', 'product'].edge_index = ...  # [2, num_edges]

# Adding reverse edges:
from torch_geometric.transforms import ToUndirected

data = ToUndirected()(data)
```

We provide a **full example** for **loading raw `*.csv` files** in the **documentation**

Heterogeneous Graph Support

Heterogeneous Graph Neural Networks

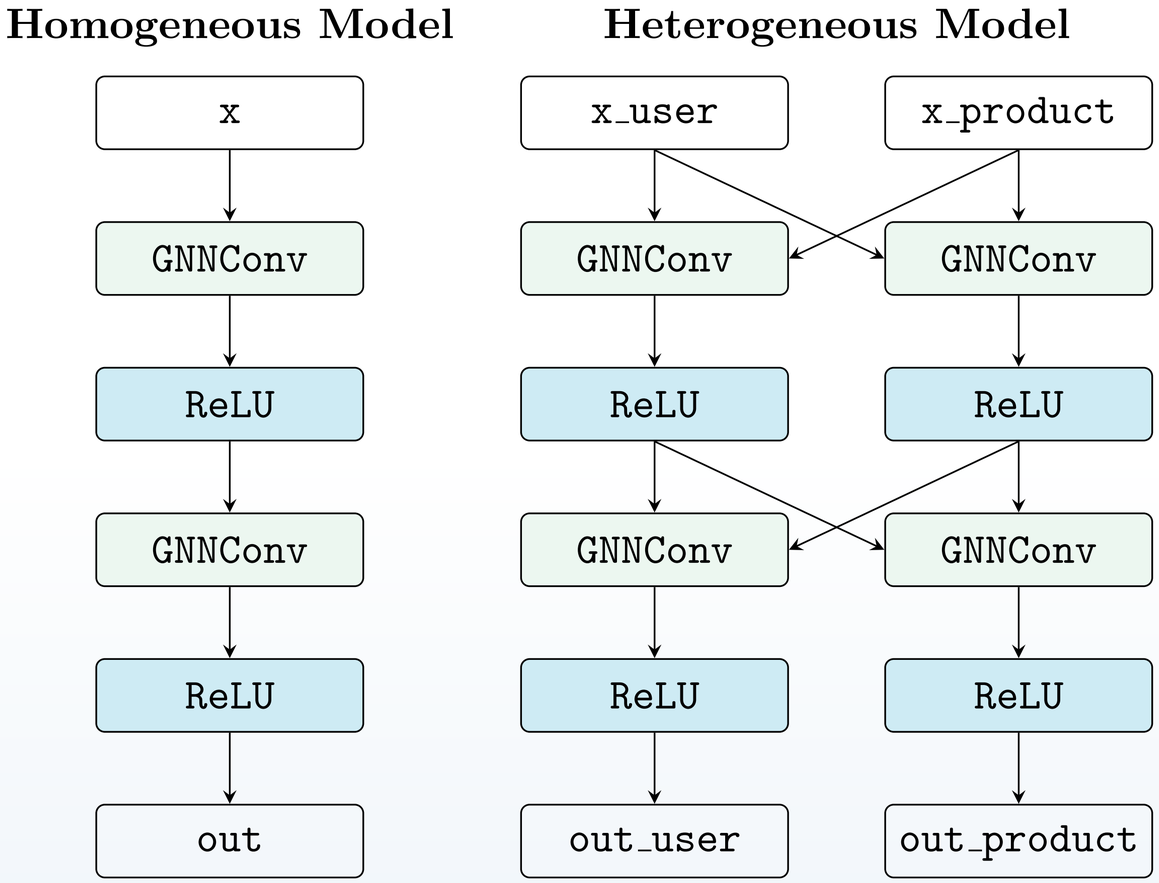
A **homogeneous GNN** can be converted to a **heterogeneous one** by learning **distinct parameters** for **each individual edge type:**

A custom GNN for each relation

$$\mathbf{h}_i^{(\ell+1)} = \sum_{r \in \mathcal{R}} \mathrm{GNN}_\theta^{(r)} \left(\mathbf{h}_i^{(\ell)}, \{ \mathbf{h}_j^{(\ell)} : j \in \mathcal{N}^{(r)}(i) \} \right)$$

The number of relations

Relational-wise neighborhood



Heterogeneous Graph Support

Heterogeneous Graph Neural Networks

Rapid growth in the number of parameters *w.r.t* number of relations may lead to overfitting on rare relations

Basis-decomposition for regularization

A custom GNN for each basis

$$\mathbf{h}_i^{(\ell+1)} = \sum_{r \in \mathcal{R}} \sum_{b=1}^{B} \mathrm{GNN}_\theta^{(b)} \left(\mathbf{h}_i^{(\ell)}, \{ a_{r,b}^{(\ell)} \cdot \mathbf{h}_j^{(\ell)} : j \in \mathcal{N}(i) \} \right)$$

The number of bases

Relational-depend trainable coefficients

Heterogeneous Graph Support

PyG can **automatically convert homogeneous GNNs to heterogeneous ones**

```
to_hetero(model (node_types, edge_types)):
to_hetero_with_bases(model (node_types, edge_types)):
```

1. **Duplicates** message passing modules for **each edge type**
2. **Transforms** the **underlying computation graph** so that messages are exchanged along **different edge types**
3. Uses **lazy initialization** (–1) to handle different **input feature dimensionalities**

```python
from torch_geometric.nn import GAT, to_hetero

model = GAT(in_channels=-1, hidden_channels=64,
            out_channels=72, num_layers=2)

model = to_hetero(model, (node_types, edge_types))

out = model(data.x_dict, data.edge_index_dict)
```

Heterogeneous Graph Support

Heterogeneous Graph Samplers

Scaling up heterogeneous GNNs
to large-scale graphs with ease
via **relational neighbor sampling**

Only requires a few lines
of code change!

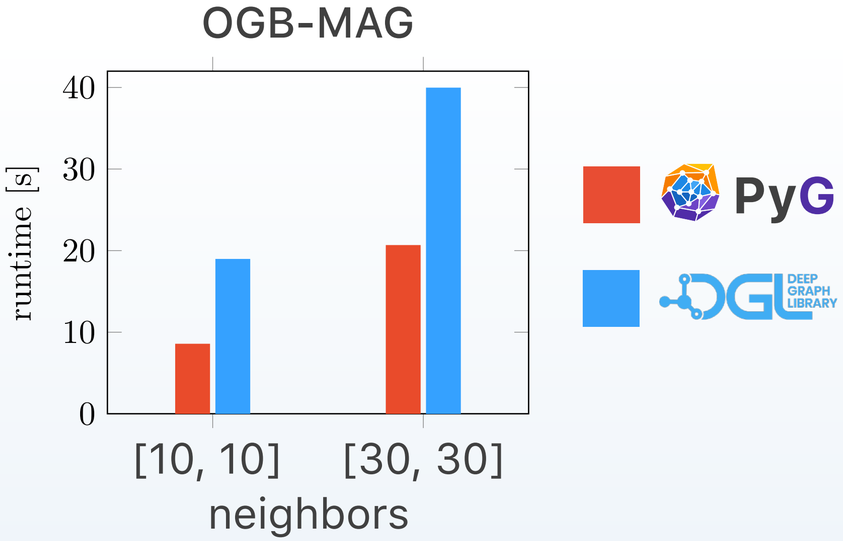
```python
from torch_geometric.datasets import OGB_MAG
from torch_geometric.loader import NeighborLoader

data = OGB_MAG(path)[0]

loader = NeighborLoader(
    data,
    num_neighbors={key: [15, 10] for key in data.edge_types},
    batch_size=128,
    input_nodes='paper')

for batch in loader:
    model(batch.x_dict, batch.edge_index_dict)
```

Heterogeneous Graph Support

Heterogeneous Graph Samplers

Scaling up heterogeneous GNNs
to large-scale graphs with ease
via **relational neighbor sampling**

Only requires a few lines of code change!




```python
from torch_geometric.datasets import OGB_MAG
from torch_geometric.loader import NeighborLoader

data = OGB_MAG(path)[0]

loader = NeighborLoader(
    data,
    num_neighbors={key: [15, 10] for key in data.edge_types},
    batch_size=128,
    input_nodes='paper')

for batch in loader:
    model(batch.x_dict, batch.edge_index_dict)
```

Heterogeneous Graph Support

- A **tutorial** introducing **all newly released heterogeneous graph features**
- A **tutorial** showcasing how to **load heterogeneous graphs** from **raw `*.csv` files**
- **Heterogeneous graph transformations**
- **Conversion** from **heterogeneous graphs** to **homogeneous "typed" graphs**
- A **generic wrapper** (`HeteroConv`) for computing heterogeneous graph convolution via **different message passing operators**
- **Lazy initialization** (`-1`) for all message passing operators in **PyG**
- A variety of **heterogeneous GNN examples**, including an example for **scaling heterogeneous graph models** via **PyTorch Lightning**
- Dedicated **heterogeneous graph operators** (`HGTConv`) and **samplers** (`HGTLoader`)

`https://pytorch-geometric.readthedocs.io`
`/pyg-team/pytorch-geometric/releases`

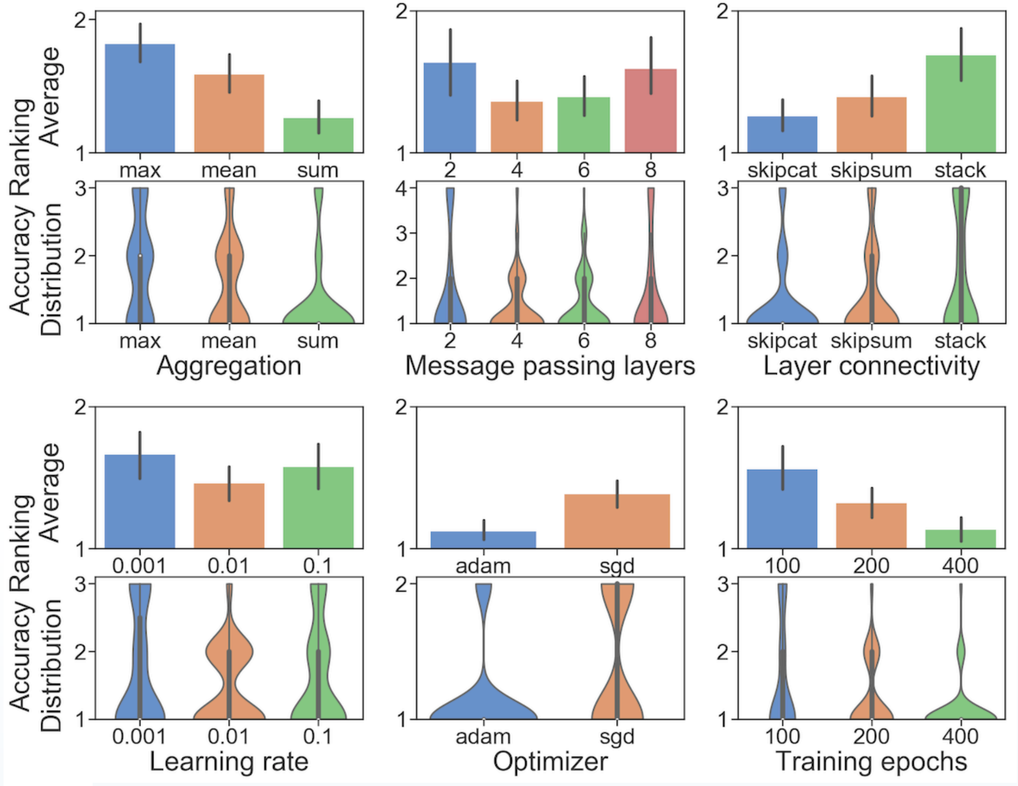
GraphGym

Design Space Exploration with GraphGym

Which GNN is the best for your given task?

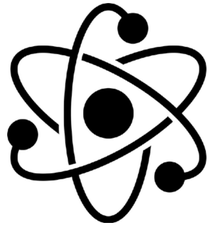
GraphGym aims to design and train GNNs from **configurations**, using a **modularized pipeline:**

- **Standardized** GNN implementation/evaluation
- **Design space exploration** via simple interfaces to try out thousands of GNN architectures **in parallel**
- Hyper-parameter search and visualizations



Future Plans

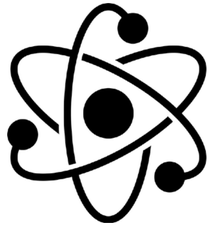
Future Plans



Temporal Graphs

- *(Nearly)* all real-world graphs are **inherently dynamic**
- Support for **addition** and **removal** of **nodes** and **edges**
- Integration of **temporal Graph Neural Networks**
- **Real-Time In-Stream Inference**

Future Plans



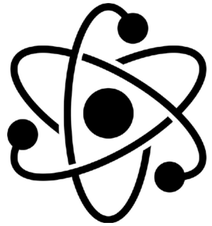
Temporal Graphs

- *(Nearly)* all real-world graphs are **inherently dynamic**
- Support for **addition** and **removal** of **nodes** and **edges**
- Integration of **temporal Graph Neural Networks**
- **Real-Time In-Stream Inference**

Distributed Data

- While distributed training is possible, **distributing data is currently a user task**
- Scaling to **billions of nodes** via distributing input data
- **Partitioning** input node and edge features

Future Plans



Temporal Graphs

- *(Nearly)* all real-world graphs are **inherently dynamic**
- Support for **addition** and **removal** of **nodes** and **edges**
- Integration of **temporal Graph Neural Networks**
- **Real-Time In-Stream Inference**

Distributed Data

- While distributed training is possible, **distributing data is currently a user task**
- Scaling to **billions of nodes** via distributing input data
- **Partitioning** input node and edge features

Auto-Scaling

- **PyG** should **automatically determine the best scalability approach** for the given task
- Write GNNs in **full-batch mode** and let **PyG** figure out the rest
- **GNNAutoScale** (ICML 2021)

Conclusion

PyG bundles the **state-of-the-art** in **Graph Representation Learning**

- ✓ 50+ **GNN architectures**
- ✓ 200+ **benchmark datasets**
- ✓ Dedicated **sparsity-aware CUDA kernels**
- ✓ **Multi-GPU** support
- ✓ **Half-Precision** support
- ✓ Support for **scalability techniques**
- ✓ **Heterogeneous graph** support
- ✓ GNN Design Space Exploration

We are constantly encouraged to make PyG even better!

dortmund university

Stanford University

team@pyg.org

https://pyg.org

/pyg-team/pytorch-geometric


```
conda install pyg -c pyg
```