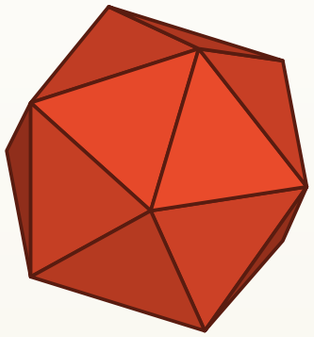
dortmund university

SFB 876 Providing Information
by Resource-Constrained Data Analysis



PyTorch geometric

Let's build a community-driven GNN platform

Matthias Fey

matthias.fey@udo.edu

/rusty1s/pytorch_geometric

Motivation

A **fast and flexible framework for GNNs** that ...

- *helps users to apply GNNs to a specific problem where relational learning is important*
- *helps researchers to invent novel methods*
- *helps researchers to compare to related work*
- *aims to make research overall more reproducible*

Motivation

We shouldn't need to write new projects completely from scratch! Instead, why not make use of well-tested ...

Datasets *Transformations* *Data Loaders*
Message Passing Interfaces *Operators*
or even *complete Models?*

In the beginning, PyTorch Geometric was meant to **accelerate my own research**, but it turned out that it can accelerate the research of others as well!

Inspiration

PyTorch Geometric is pure PyTorch with GNN support

Model Definition

```
class MLP(torch.nn.Module):
  def __init__(self):
    super().__init__()
    self.lin1 = Linear(x, y)
    self.lin2 = Linear(y, z)

def forward(self, x):
  # x: [batch_size, #features]


  x = self.lin1(x)
  x = x.relu()
  x = self.lin2(x)
  return x
```

```
class GNN(torch.nn.Module):
  def __init__(self):
    super().__init__()
~   self.conv1 = GCNConv(x, y)
~   self.conv2 = GCNConv(y, z)

~ def forward(self, x, edge_index):
    # x: [num_nodes, #features]
    # edge_index: [2, num_edges]

~   x = self.conv1(x, edge_index)
    x = x.relu()
~   x = self.conv2(x, edge_index)
    return x
```

Inspiration

PyTorch Geometric is **pure PyTorch** with **GNN support**

Dataset Initialization

```
dataset = MNIST(path, train=True,
                transform=T.Compose([T.ColorJitter(), T.ToTensor()])
loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

```
~ dataset = MoleculeNet(path, name='PCBA',
~                       transform=T.Compose([T.ToUndirected(), T.AddSelfLoops()])
  loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

Inspiration

PyTorch Geometric is **pure PyTorch** with **GNN support**

Training Procedure

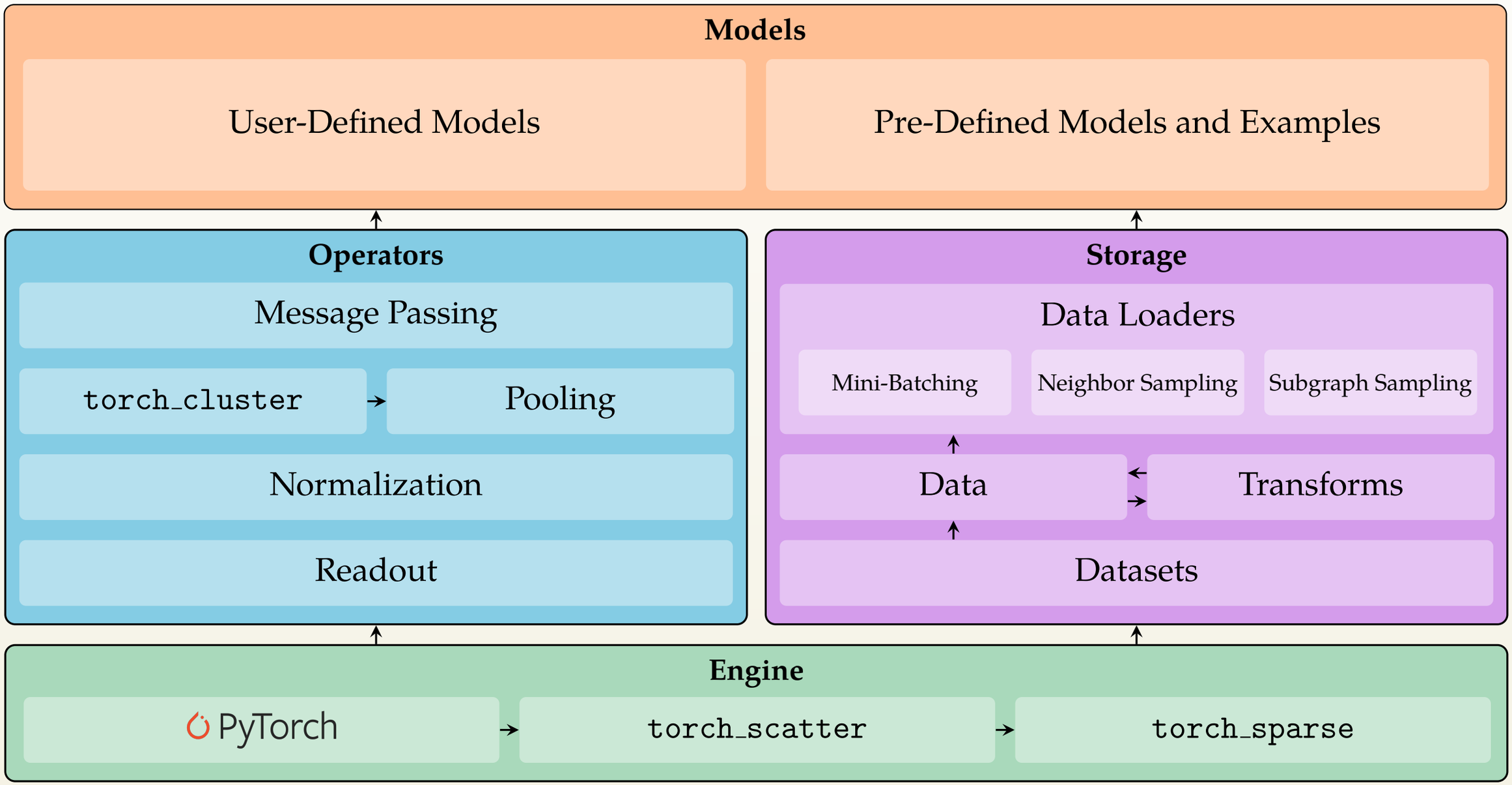
```
for x, y in loader:
    x, y = x.cuda(), y.cuda()
    out = model(x)
    loss = criterion(out, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
~ for data in loader:
~     data = data.cuda()
~     out = model(data.x, data.edge_index)
~     loss = criterion(out, data.y)
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
```

Overview

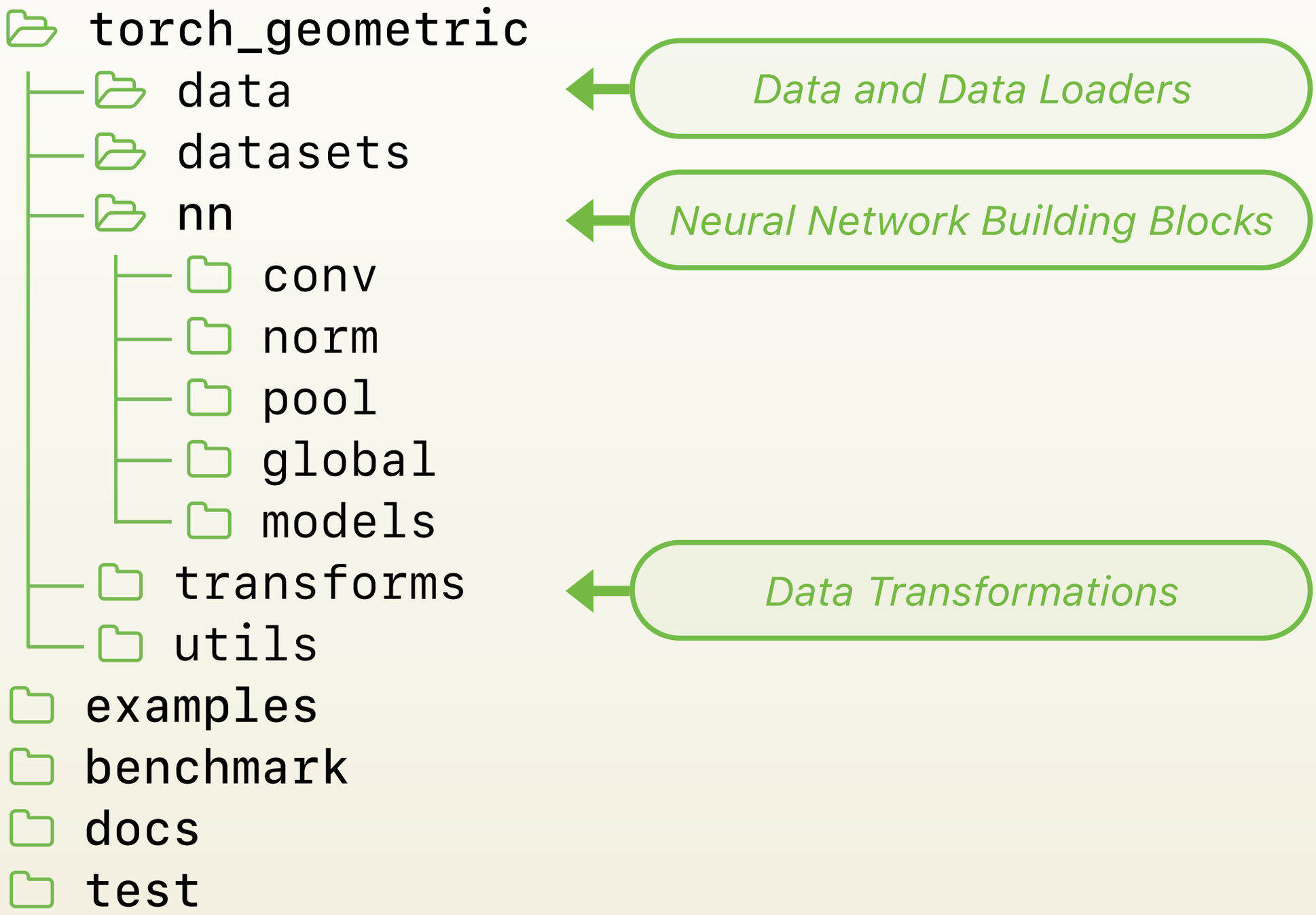
GNNs	Cheby GCN SAGE PointNet MoNet MPNN GAT SplineCNN AGNN EdgeCNN S-GCN R-GCN PointCNN ARMA APPNP GIN GIN-E CG GatedGCN NMF TAG Signed-GCN DNA PPFNet FeaST Hyper-GCN GravNet ResGatedGCN SparseTransformer DGCNN FeaST EG LeCNN PNA GEN GCN2 PAN FiLM SuperGAT FA
Norm	GraphNorm GraphSizeNorm PairNorm DiffGroupNorm
Pooling	Set2Set SortPool DiffPool MinCUT SAG Graclus VoxelGrid TopK EdgePool ASAP PAN MemPool FPS
Models	(V)GAE ARG(V)A DGI Node2Vec Graph-UNet AFP C&S LP GeniePath SchNet DimeNet MetaPath2Vec RENet TGN SEAL
Utilities	GNNExplainer SIGN GDC ClusterGCN DropEdge JK GraphSAINT NeighborSampling ShaDow LDP

Overview: Architecture



Overview: File Structure

PyTorch Geometric



Overview: File Structure

PyTorch Geometric

- torch_geometric
  - data
  - datasets
  - nn
    - conv
    - norm
    - pool
    - global
    - models
  - transforms
  - utils
- examples
- benchmark
- docs
- test

PyTorch Scatter

- csrc
  - cpu
  - cuda
- torch_scatter
- docs
- test

Low-level C++ and CUDA Implementations

Python Bindings

Recent Highlights: Large-Scale Graphs

```python
def train(data):  # Full-batch training
    out = model(data.x, data.edge_index)
    loss = criterion(out, data.y)
```

RuntimeError: CUDA error: out of memory

```python
+ loader = ClusterLoader(  # Cluster-GCN
+     ClusterData(data, num_parts=128), batch_size=32)

+ loader = GraphSAINTRandomWalkSampler(  # GraphSAINT
+     data, batch_size=128, walk_length=3)

~ def train(loader):  # Mini-batch training
+     for data in loader:
          out = model(data.x, data.edge_index)
          loss = criterion(out, data.y)
```

Recent Highlights: SparseTensor

```python
from torch_sparse import SparseTensor

adj = SparseTensor(row=edge_index[0], col=edge_index[1],
                   value, sparse_sizes=(num_nodes, num_nodes))
# value is optional and can be None

# Obtain different representations (COO, CSR, CSC):
row,    col, value = adj.coo()
rowptr, col, value = adj.csr()
colptr, row, value = adj.csc()

adj = adj[:100, :100]  # Slicing, indexing and masking support
adj = adj.set_diag()   # Add diagonal entries
adj = adj.t()          # Transpose
out = adj.matmul(x)    # Sparse-dense matrix multiplication
adj = adj.matmul(adj)  # Sparse-sparse matrix multiplication
```

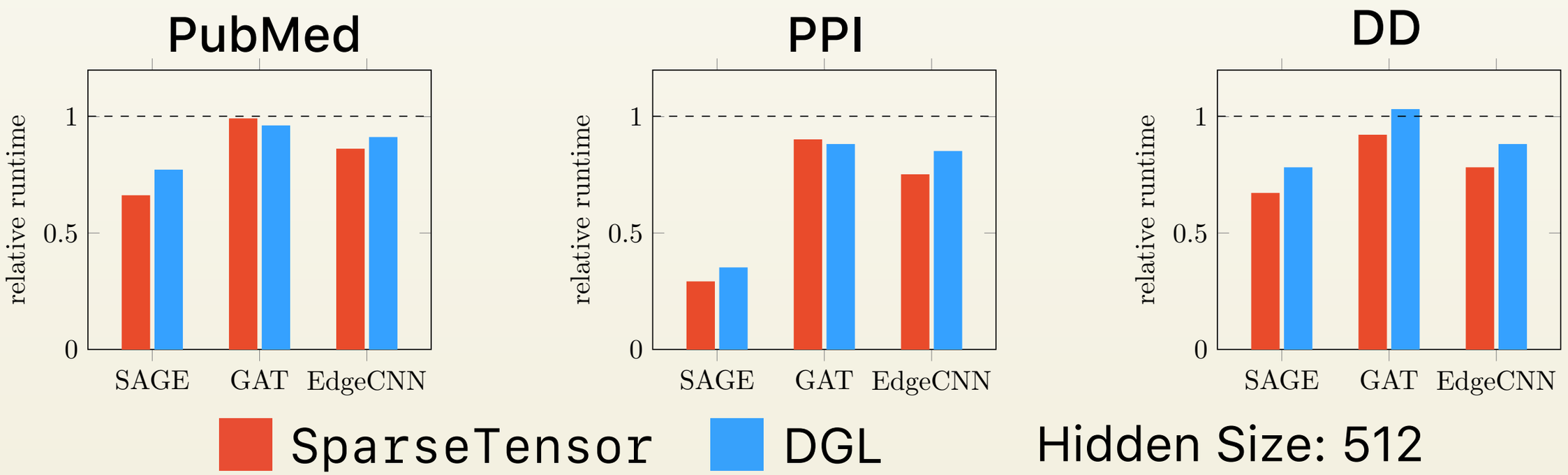
Recent Highlights: Memory-Efficiency

Utilize sparse matrix multiplication in message passing

```
class GINConv(MessagePassing):
  def forward(self, x, edge_index: Union[Tensor, SparseTensor]):
    out = self.propagate(edge_index, x=x)
    return MLP((1 + eps) * x + out)
```

```
+  def message_and_aggregate(self, adj: SparseTensor, x):
+    return adj.t().matmul(x, reduce='sum')
```

Custom aggregations



Recent Highlights: PyTorch Lightning

```
+  class Dataset(LightningDataModule):
+      def train_dataloader(self):
+          return NeighborSampler(self.data, ...)

~  class GNN(LightningModule):
       def forward(self, ...):
           ...

+      def training_step(self, batch):
+          out = self(batch.x, batch.edge_index)
+          loss = criterion(out, batch.y)
+          return loss

   # Out-of-the-box Distributed Data Parallel Support
+  trainer = Trainer(gpus=2, accelerator='ddp', max_epochs=10)
+  trainer.fit(model, datamodule)
```

Recent Highlights: TorchScript

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

       def forward(self, x, edge_index):
           x = self.conv1(x, edge_index)
           x = x.relu()
           x = self.conv2(x, edge_index)
           return x


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

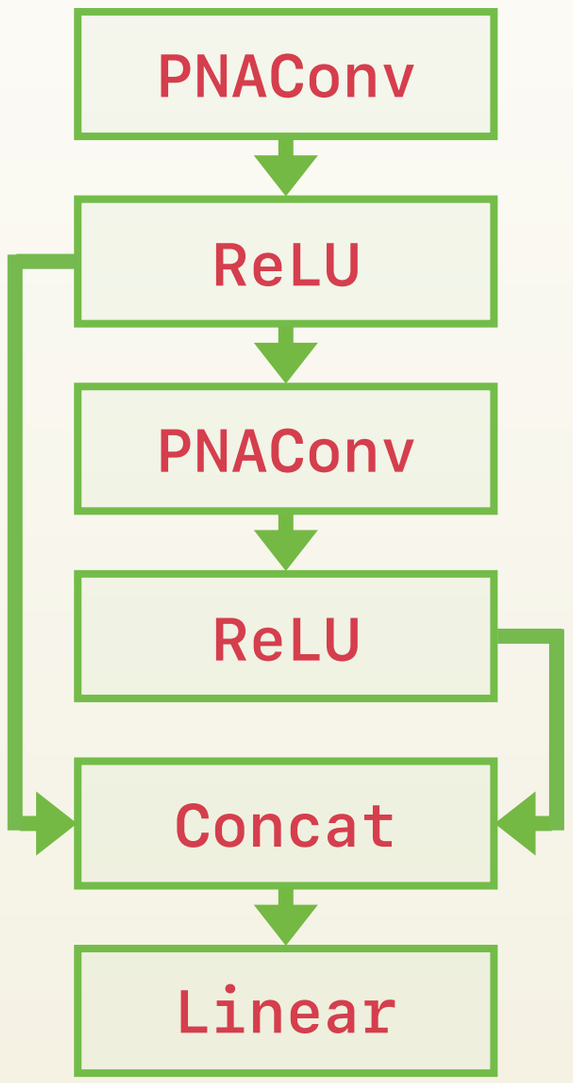
Create "jittable" instances

Create TorchScript program

Recent Highlights: Sequential

Fast and intuitive **sequential GNN construction**

```
model = Sequential('x, edge_index',  # Model arguments
  [
    (PNAConv(in_channels, 64), 'x, edge_index -> x1'),

     ReLU(),

    (PNAConv(64, 64), 'x1, edge_index -> x2'),

     ReLU(),

    (lambda x1, x2: torch.cat([x1, x2]), 'x1, x2 -> x'),

     Linear(128, out_channels),
  ]
)
```



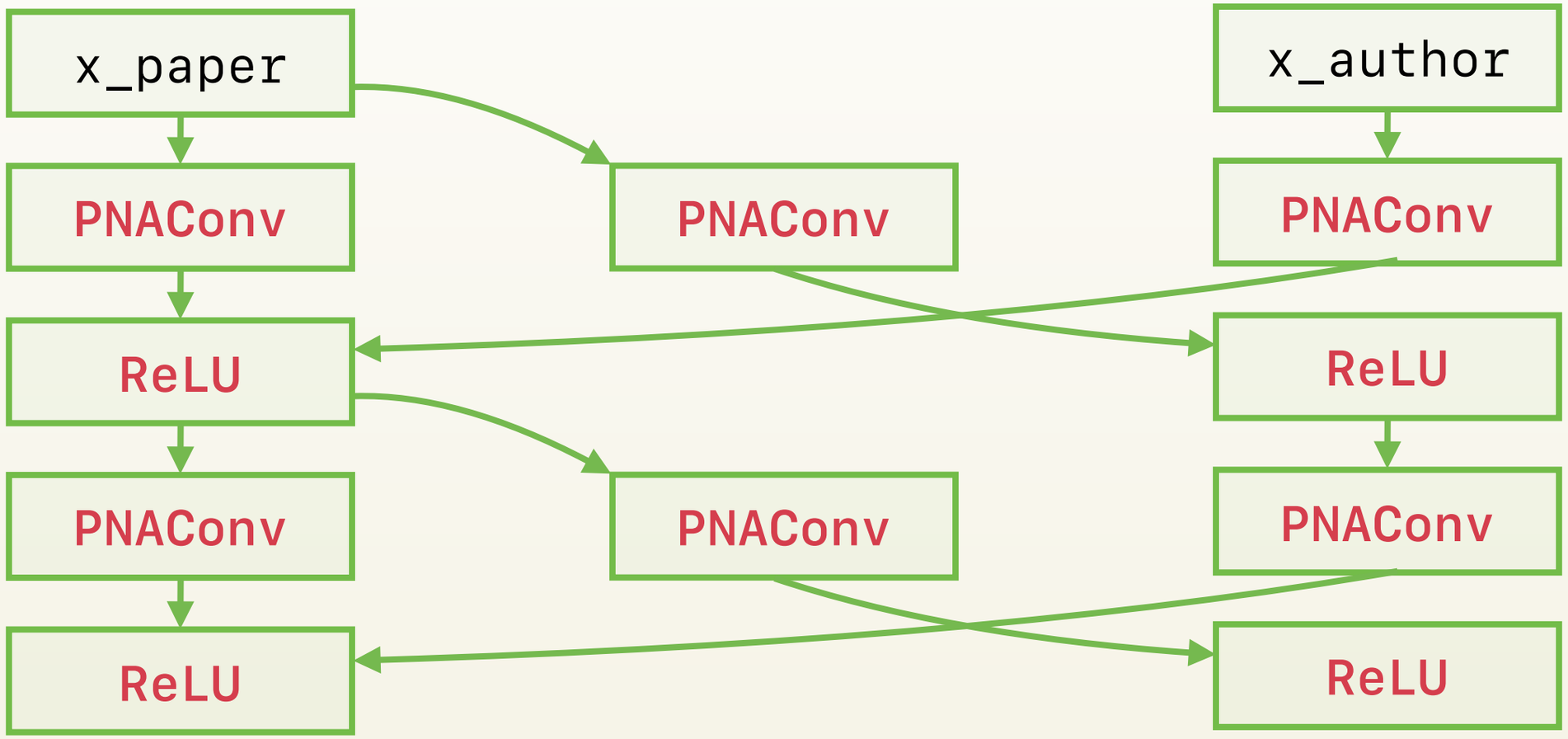
Future Plans: Heterogeneous Graphs

First-class **Heterogeneous Graph Storage** Support

```
data = HeteroData()
data['paper'].x = ...
data['author'].x = ...
data['paper', 'cites', 'paper'].edge_index = ...
data['author', 'writes', 'paper'].edge_index = ...


# Add reverse edges:
data = T.ToUndirected()(data)
```

Future Plans: Heterogeneous Graphs

First-class **Heterogeneous Graph Processing** Support



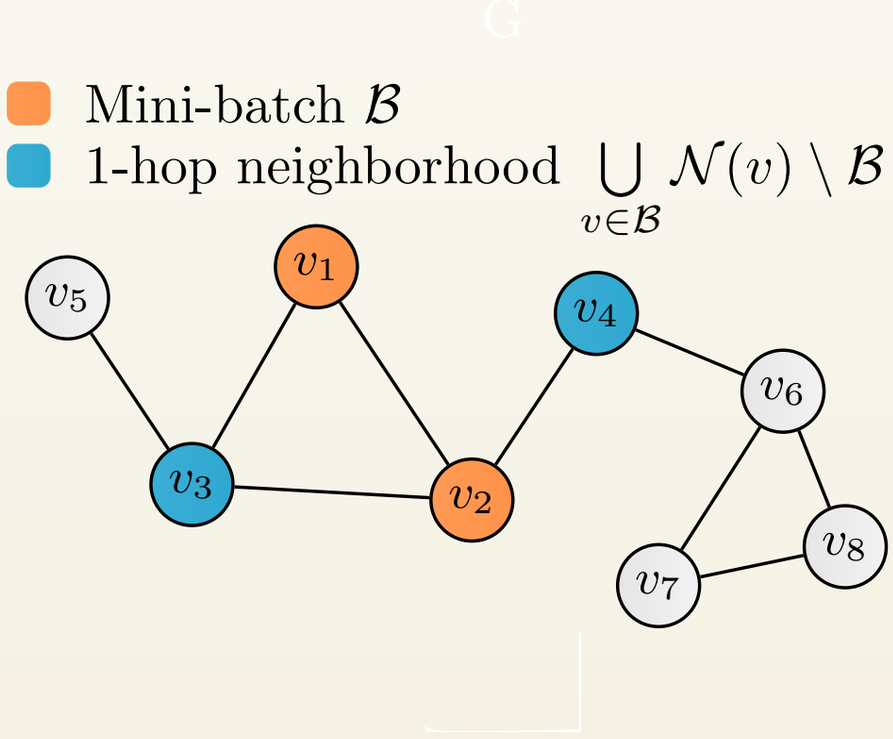

```
model = GNN()
hetero_model = to_hetero(model, aggr='sum')
```

Future Plans: PyG-AutoScale (PyGAS)

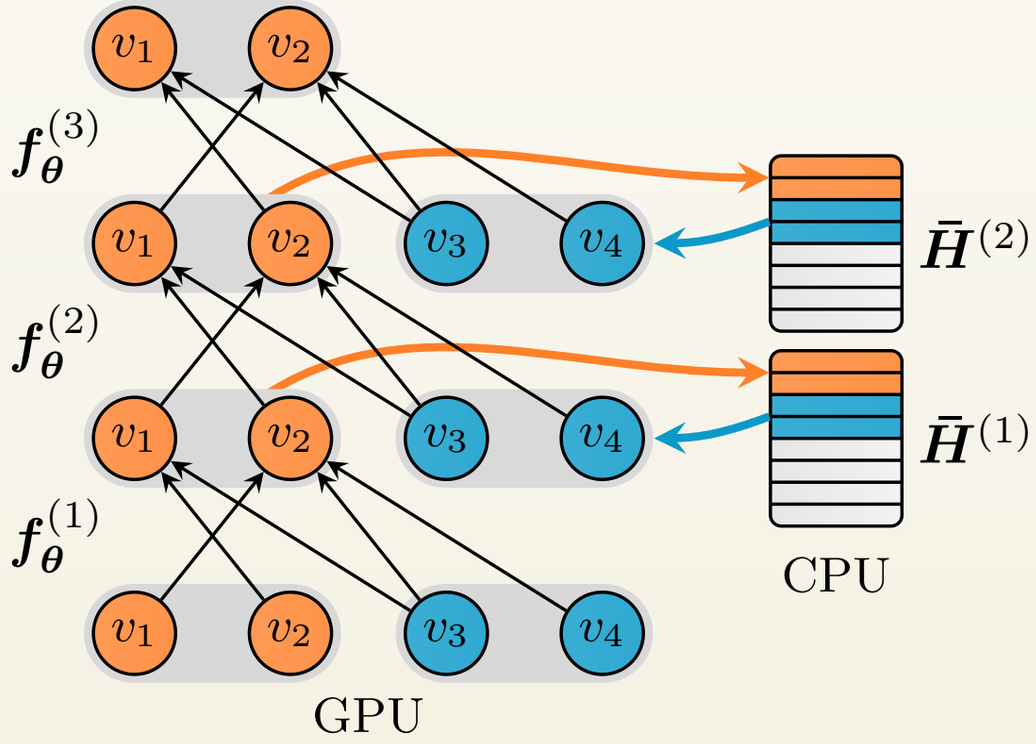
An extension to PyG that converts any GNN model into its **scalable variant**

Fey, Lenssen, Weichert, Leskovec (ICML 2021)

GNNAutoScale: *Scalable and Expressive Graph Neural Networks via Historical Embeddings*



Mini-batch selection



Computation graph

Future Plans: PyG-AutoScale (PyGAS)

An extension to PyG that converts any GNN model into its **scalable variant**

```
~ class GNN(ScalableGNN):
    def __init__(self, ...):
~       super().__init__(num_nodes, hidden_channels, num_layers)
        self.conv1 = GCNConv(...)
        self.conv2 = GCNConv(...)

~   def forward(self, x, edge_index, node_idx):
        x = self.conv1(x, edge_index)
+       x = self.push_and_pull(self.histories[0], x, node_idx)
        x = x.relu()
        x = self.conv2(x, edge_index)
        return x
```

Contributing

PyTorch Geometric is a **community-driven effort:**

- *We rely on contributions to keep up with integrating recent research advancements*

We are trying to **build a community:**

- *Feel free to join our Slack channel* 

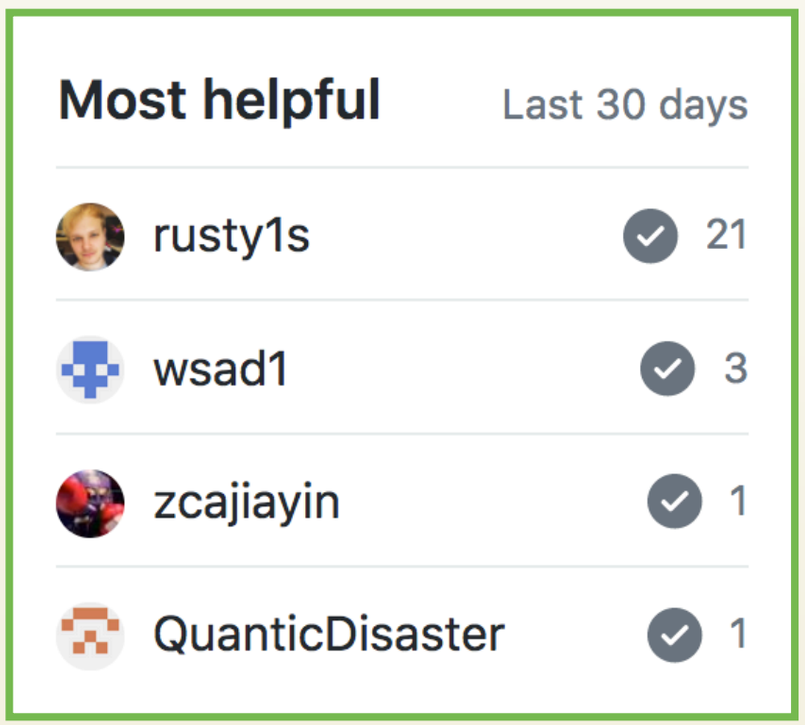
If you are interested in **actively contributing**, please reach out to me so we can join forces

Thanks to many wonderful contributors!

Contributing

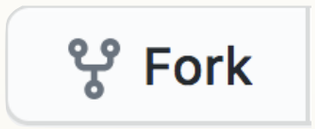
You can contribute in so many ways!

- *Implementing new layers, models and examples*
- *Answering questions and helping others*
- *Improving documentation*
- *Submitting issues for bugs or new features*
- *Reviewing Pull Requests*



Developing PyTorch Geometric

1. **Fork** PyTorch Geometric and **clone** it:


   ```
   git clone https://github.com/*/pytorch_geometric
   ```

2. **Install** PyTorch Geometric in **develop mode**:

   ```
   python setup.py develop
   ```

3. Apply modifications

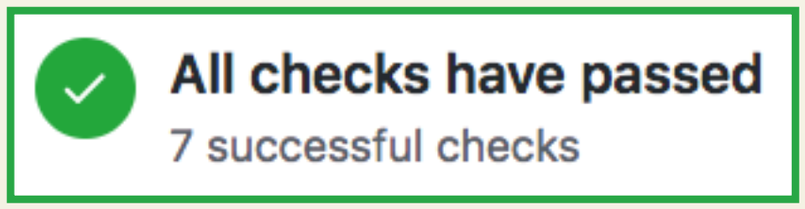
4. Ensure that **all tests pass**:

   ```
   python setup.py test
   ```

5. Create a **Pull Request**:

6. Ensure that **all CI checks pass**:

Longevity of PyTorch Geometric

We are building a larger team for the future development of PyTorch Geometric, in close collaboration with

dortmund university

Stanford University

- ✓ ensure longevity of PyTorch Geometric
- ✓ keep up with integrating the latest research
- ✓ extend its scope
- ✓ make it easier to use for both academia and industry
- ✓ make it more scalable