GNN Implementation: PyG & GraphGym

Jiaxuan You

Assistant Professor at UIUC CDS

CS598: Deep Learning with Graphs, 2024 Fall

https://ulab-uiuc.github.io/CS598/

Logistics: Proposal Task Due

- The deadline for submission is **Oct 6 (Sunday), 11:59 PM, CT**.
- Please submit a **PDF** file to Canvas for each group. Each group only needs to submit once. Only the last submission from each group will be graded.
- You are required to use the ICLR 2025 template. **Abstracts are not required.** Please contact the TA if you have trouble using LaTeX.
- Submissions are suggested to be between 1.5 to 2 pages in length, with a minimum of 1 page, excluding references.

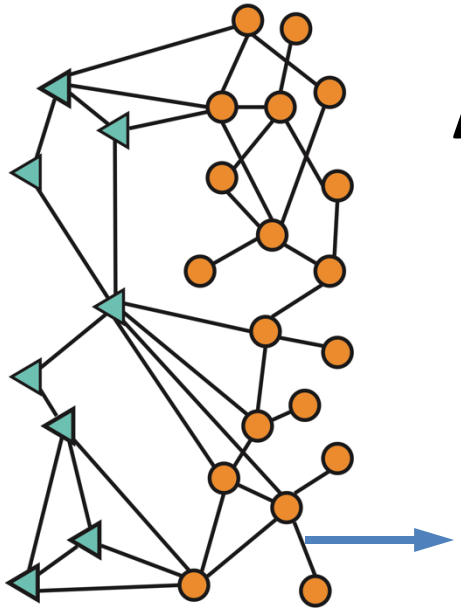
Logistics: Project Idea Discussion

- **Wednesday Schedule**
  - Each group is required to present your proposal **(around 3 minutes)** and answer questions from instructor and classmates **(around 1 minute)**.
  - Slides are **NOT** required but are welcomed.
  - Share your proposal in Slack and provide your feedback to others' proposal after class.
    - As part of grading for "Ideate and discussion", peer discussion on Slack will count towards **2%** of your final grade. Please provide feedback to **at least 2 groups**.
  - **Attendance to each discussion session will count towards 1% of your final grade.**

GNN Implementation

GNN Design Space & GraphGym

How to Design GNNs for a New Task?



A novel GNN application

J.P.Morgan

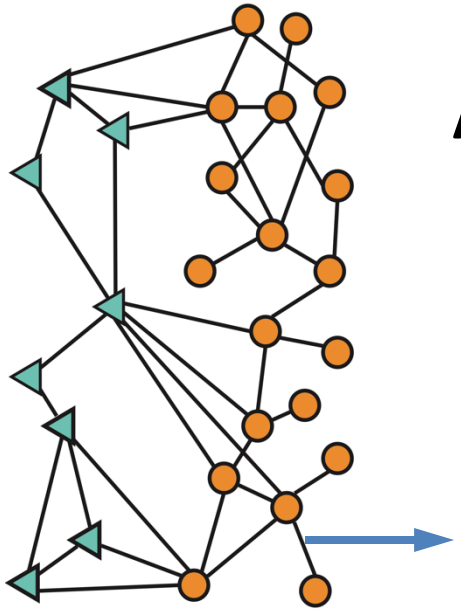
Edge-level prediction

Is this transaction fraudulent?

	ROC AUC
Non-graph baseline	0.8
GNN (out-of-the-box)	0.81

Graph ML seems to have no benefits

How to Design GNNs for a New Task?



A novel GNN application

J.P.Morgan

Edge-level prediction

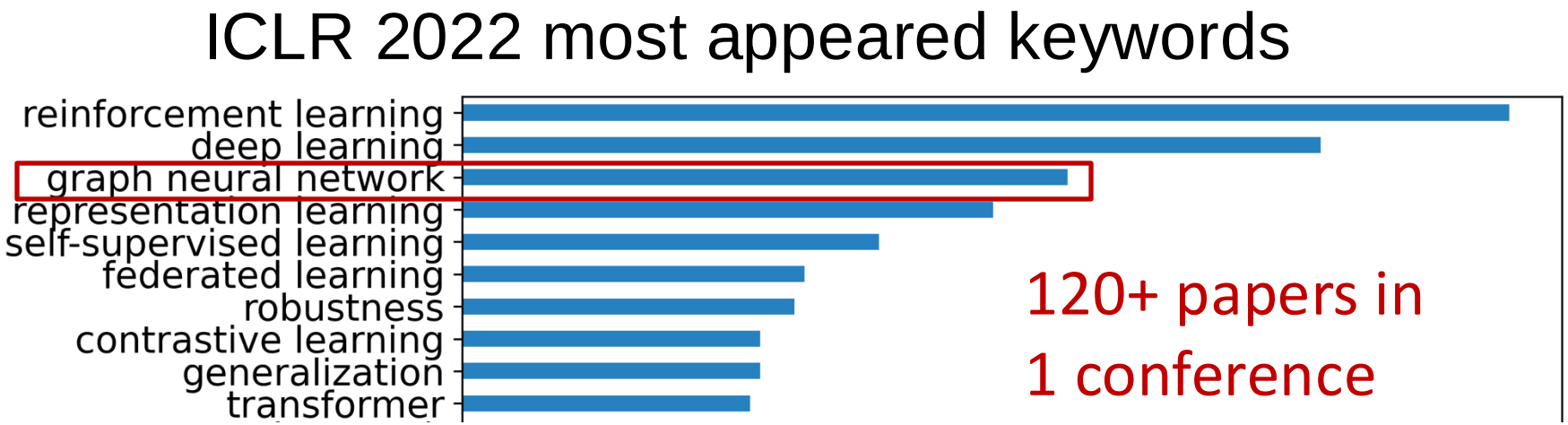
Is this transaction fraudulent?

	ROC AUC
Non-graph baseline	0.8
GNN (out-of-the-box)	0.81
GNN (careful design)	0.94

- **Question: Principled way** to design a good GNN for a new task?
  - **Promote** the **research & application** of ML on graphs

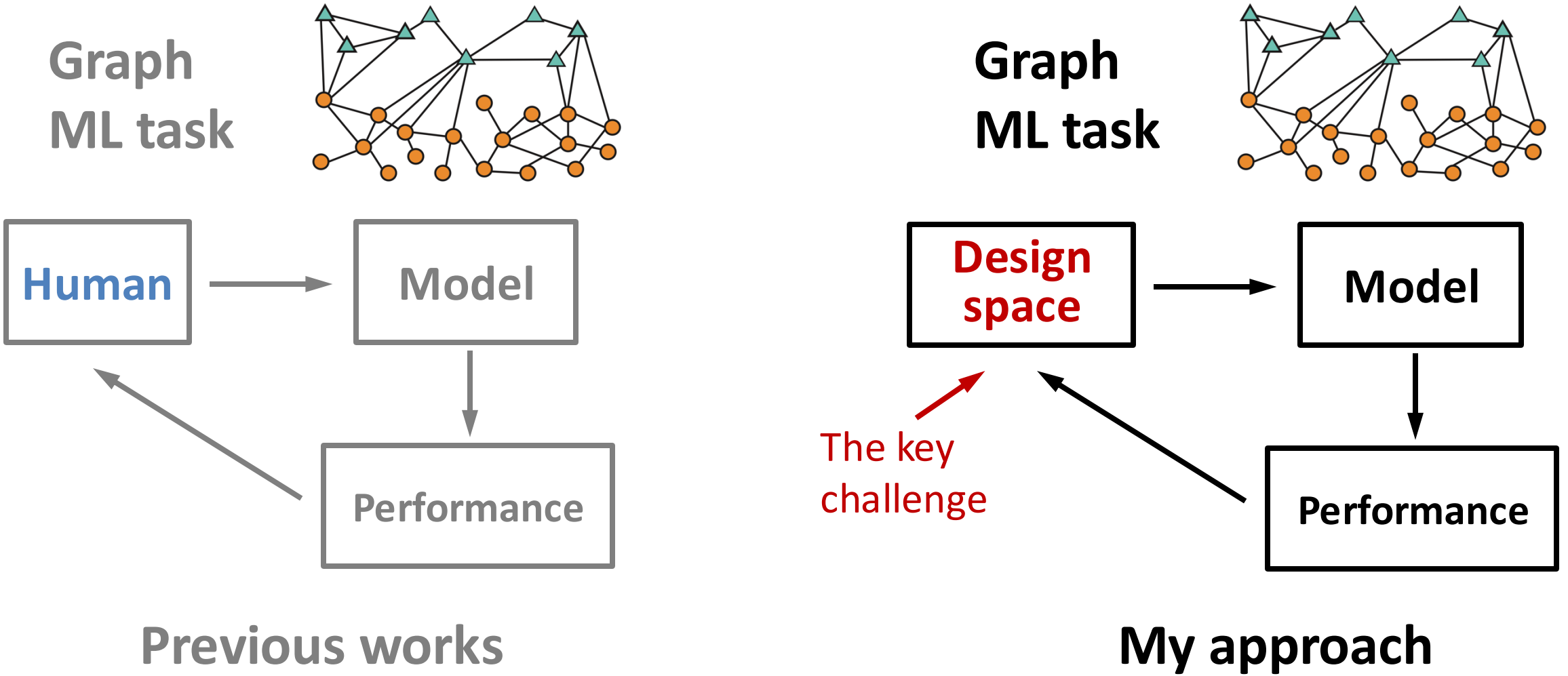
Designing GNNs is **Challenging**

- **Numerous GNN models** have been proposed – which one to use?



- **Issue: no principled approach** for finding the best GNN designs

Solution: AutoML on graphs



Background: Terminology

- **Design:** a concrete model instantiation
  - E.g., a 4-layer GraphSAGE
- **Design dimensions** characterize a design
  - E.g., the number of layers $L \in \{2, 4, 6, 8\}$
- **Design choice** is the actual selected value in the design dimension
  - E.g., the number of layers $L = 2$
- **Design space** consists of a Cartesian product of design dimensions
- **Task:** A specific task of interest
  - E.g., node classification on Cora, graph classification on ENZYMES
- **Task space** consists of all the tasks we care about

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020

Key Questions for GNN Design

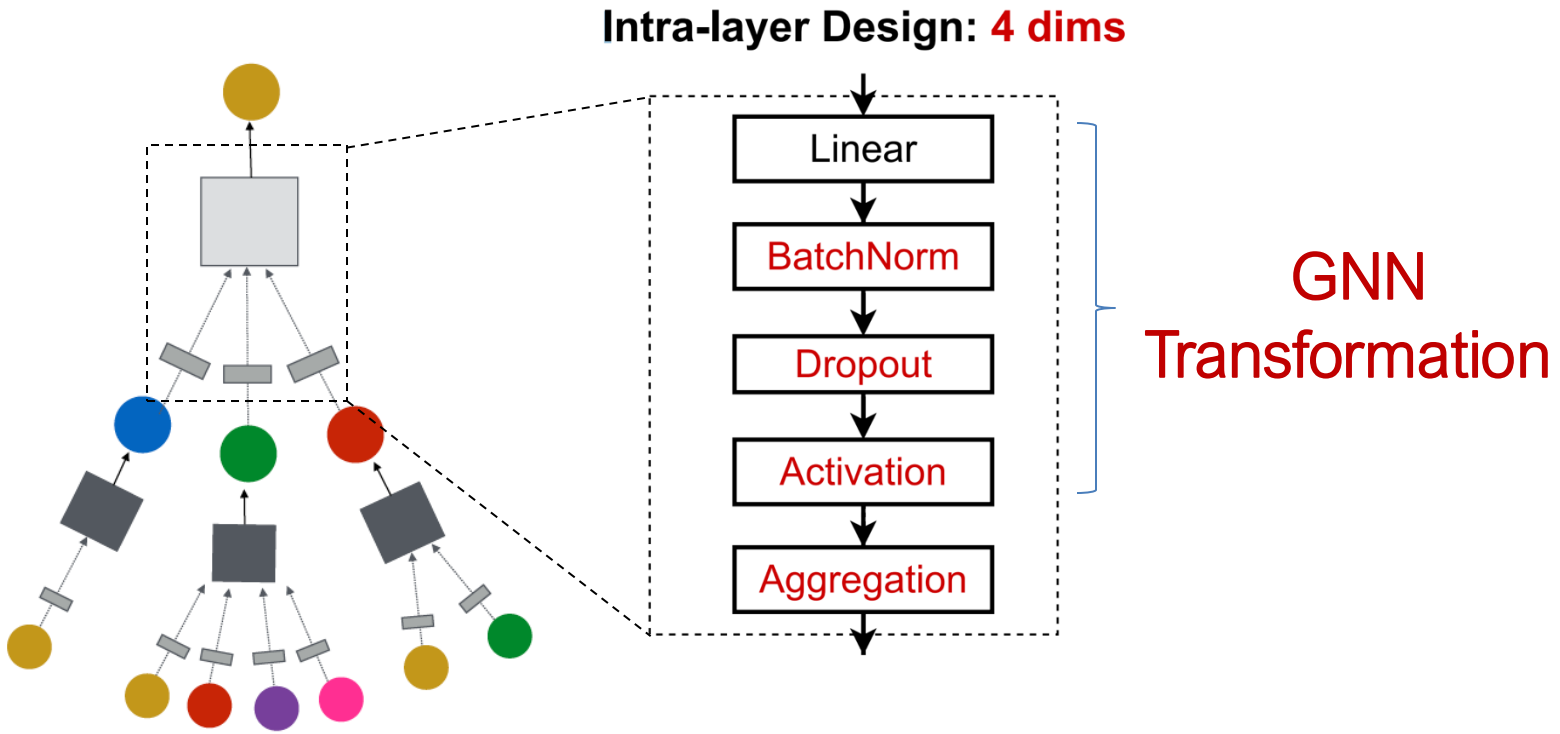
- **GNN architectural design:**
  - How to find a good GNN design for a specific GNN task?
- **Important but challenging:**
  - Domain experts want to use SOTA GNN on their specific tasks, however...
    - There are tons of possible GNN architectures
      - GCN, GraphSAGE, GAT, GIN, …
    - **Issue:** Best design in one task can perform badly for another task
    - Redo hyperparameter grid search for each new task is NOT feasible
- **Key contribution of GraphGym paper:**
  - The first systematic study for the ***GNN design space and task space***
  - **GraphGym**, a powerful platform for exploring different GNN designs and tasks

Recap: GNN Design Space

Intra-layer Design:

GNN Layer = Transformation + Aggregation

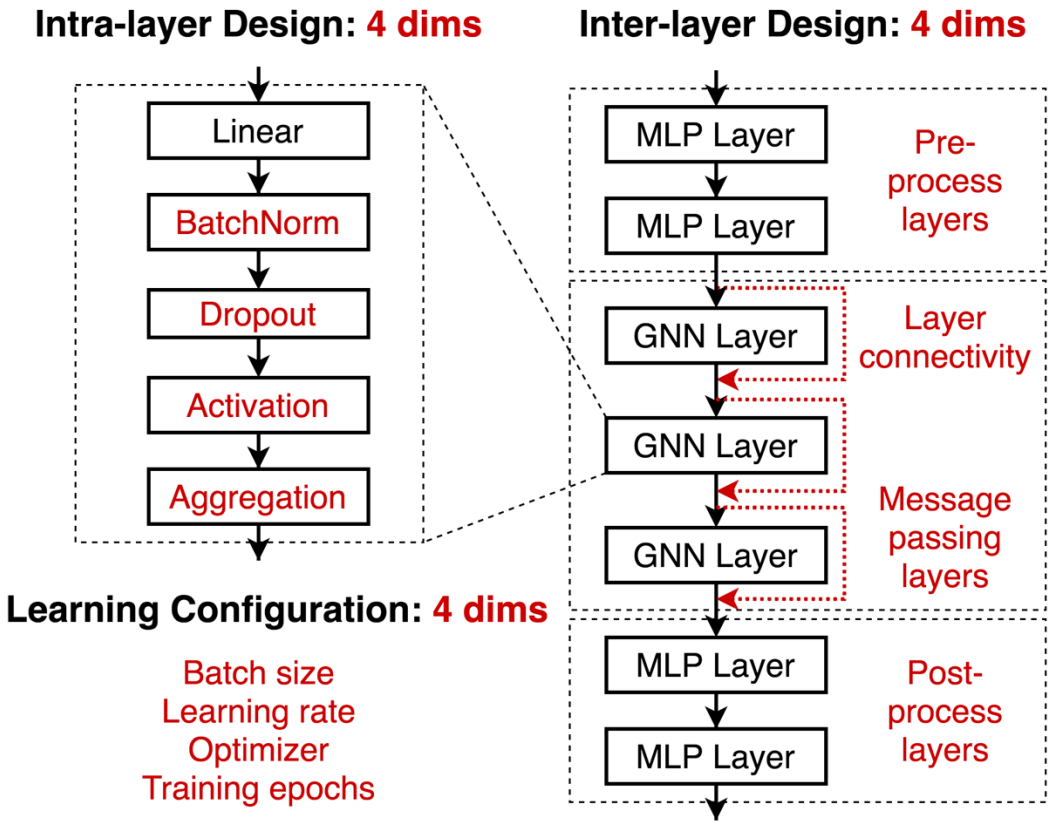
- **We propose a general instantiation under this perspective**



Recap: GNN Design Space

Inter-layer Design

- **We explore different ways of organizing GNN layers**

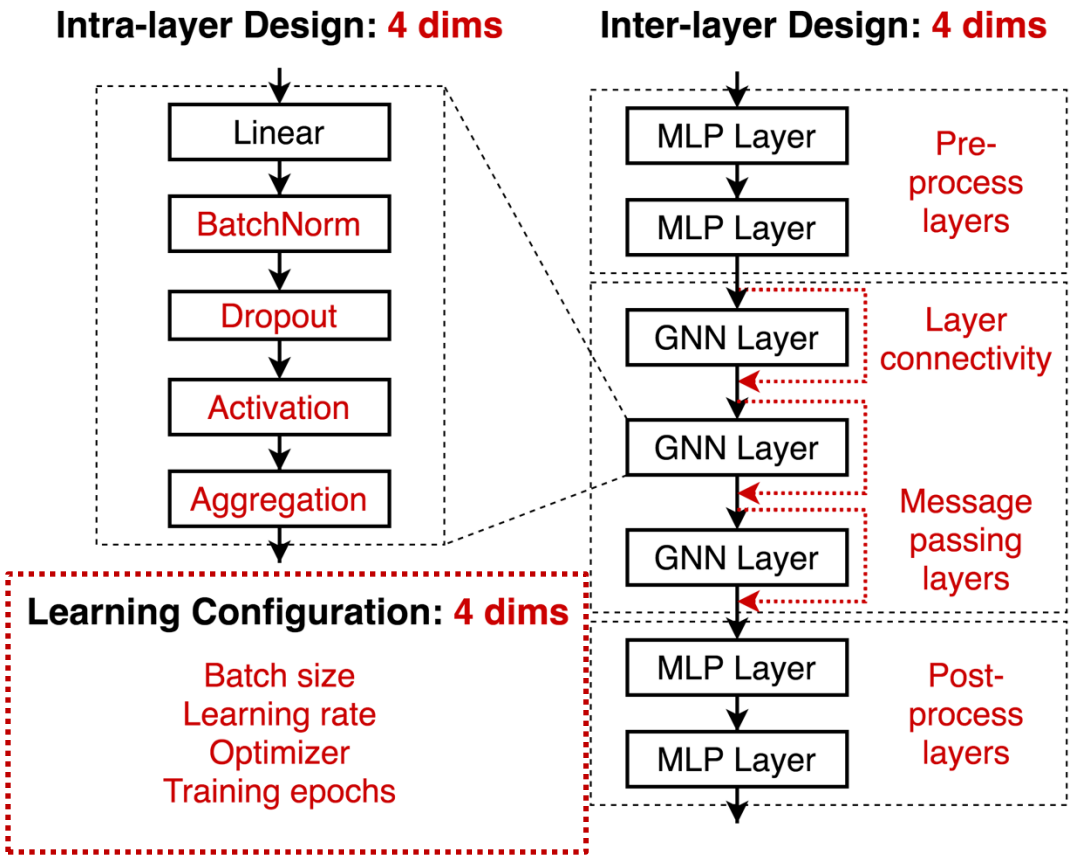


Pre-process layers:
Important when expressive node feature encoder is needed
E.g., when nodes are images/text

Skip connections:
Improve deep GNN's performance

Post-process layers:
Important when reasoning or transformation over node embeddings are needed
E.g., graph classification, knowledge graphs

Recap: GNN Design Space



Learning configurations

- **Often neglected in current literature**
- **But we found they have high impact on performance**

GNN Design Space

- **Overall: A GNN design space**
  - Intra-layer design

Batch Normalization	Dropout	Activation	Aggregation
True, False	False, 0.3, 0.6	RELU, PRELU, SWISH	MEAN, MAX, SUM

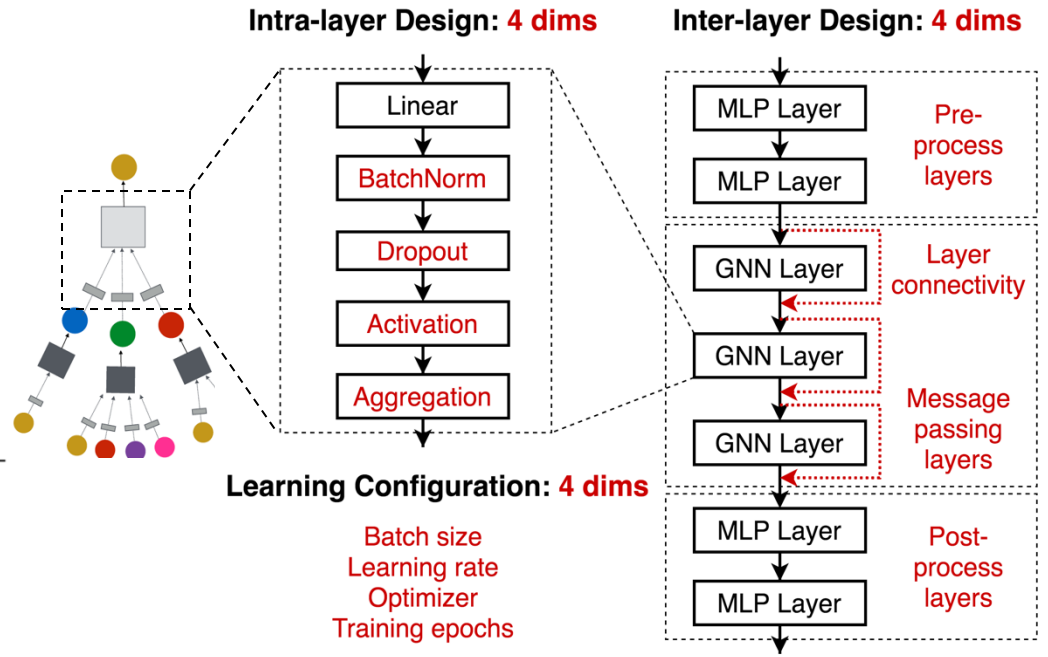
  - Inter-layer design

Layer connectivity	Pre-process layers	Message passing layers	Post-precess layers
STACK, SKIP-SUM, SKIP-CAT	1, 2, 3	2, 4, 6, 8	1, 2, 3

  - Learning configuration

Batch size	Learning rate	Optimizer	Training epochs
16, 32, 64	0.1, 0.01, 0.001	SGD, ADAM	100, 200, 400

  - **In total:** 315K possible designs



- **Purpose:**
  - We don't want to (and we cannot) cover all the possible designs
  - **A mindset transition:** We want to demonstrate that **studying a design space is more effective than studying individual GNN designs**

A General GNN Task Space

- **Categorizing GNN tasks**
  - **Common practice**: node / edge / graph level task
  - Reasonable but not precise
    - **Node prediction:** predict clustering coefficient vs. predict a node's subject area in a citation networks – **completely different task**
  - But creating a precise taxonomy of GNN tasks is very hard!
    - **Subjective**; **Novel GNN tasks** can always emerge
- **Our innovation: a quantitative task similarity metric**
  - **Purpose: understand GNN tasks, transfer the best GNN models across tasks**

A General GNN Task Space

- **Innovation:** quantitative **task similarity metric**
  - **1)** Select **"anchor" models** $(M_1, \dots, M_5)$
  - **2)** Characterize a task by ranking the performance of anchor models
  - **3)** Tasks with similar rankings are considered as similar

Task Similarity Metric

	Anchor Model Performance ranking					Similarity to Task A
Task A	M_1	M_2	M_3	M_4	M_5	1.0
Task B	M_1	M_3	M_2	M_4	M_5	0.8
Task C	M_5	M_1	M_4	M_3	M_2	-0.4

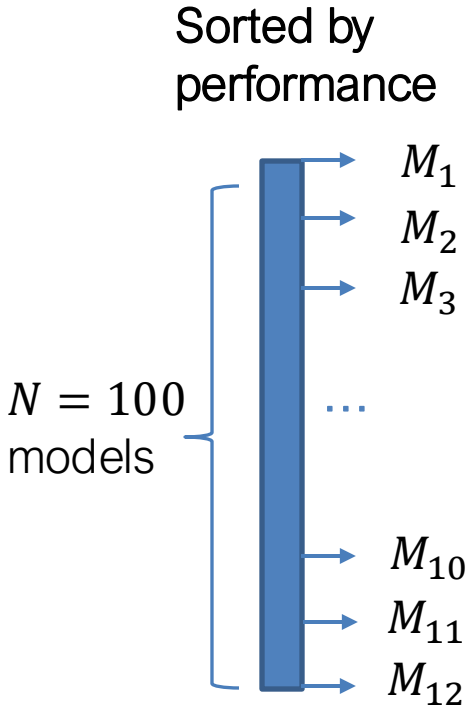
Task A is similar to Task B

Task A is not similar to Task C

- **How do we select the anchor models?**

A General GNN Task Space

- **Selecting the anchor models**
  - 1) Select a small dataset
    - E.g., node classification on Cora
  - 2) Randomly **sample N models from our design space**, run on the dataset
    - E.g., we sample 100 models
  - 3) Sort these models based on their performance: **evenly select M models as the anchor models**, whose **performance range from the worst to the best**
    - E.g., we sample 12 models in our experiments
  - **Goal: Cover a wide spectrum of models:** A bad model in one task could be great for another task



A General GNN Task Space

- **We collect 32 tasks**: node / graph classification

Task name
node-AmazonComputers-N/A-N/A
node-AmazonPhoto-N/A-N/A
node-CiteSeer-N/A-N/A
node-CoauthorCS-N/A-N/A
node-CoauthorPhysics-N/A-N/A
node-Cora-N/A-N/A
node-scalefree-clustering-pagerank
node-scalefree-const-clustering
node-scalefree-const-pagerank
node-scalefree-onehot-clustering
node-scalefree-onehot-pagerank
node-scalefree-pagerank-clustering
node-smallworld-clustering-pagerank
node-smallworld-const-clustering
node-smallworld-const-pagerank
node-smallworld-onehot-clustering
node-smallworld-onehot-pagerank
node-smallworld-pagerank-clustering
graph-PROTEINS-N/A-N/A
graph-BZR-N/A-N/A
graph-COX2-N/A-N/A
graph-DD-N/A-N/A
graph-ENZYMES-N/A-N/A
graph-IMDB-N/A-N/A
graph-scalefree-clustering-path
graph-scalefree-const-path
graph-scalefree-onehot-path
graph-scalefree-pagerank-path
graph-smallworld-clustering-path
graph-smallworld-const-path
graph-smallworld-onehot-path
graph-smallworld-pagerank-path
graph-ogbg-molhiv-N/A-N/A

6 Real-world node classification tasks

12 Synthetic node classification tasks

Predict node properties:

- Clustering coefficient
- PageRank

6 Real-world graph classification tasks

8 Synthetic graph classification tasks

Predict graph properties:

- Average path length

Evaluating GNN Designs

- **Evaluating a design dimension**:
  - "Is BatchNorm generally useful for GNNs?"
- **The common practice:**
  - **(1) Pick one model** (e.g., a 5-layer 64-dim GCN)
  - **(2) Compare two models**, with BN = True / False
- **Our approach:**
  - Note that **we have defined** $315\text{K}\ (\text{models}) * 32\ (\text{tasks}) \approx$ **10M model-task combinations**
  - **(1) Sample from 10M possible model-task combinations**
  - **(2) Rank the models** with BN = True / False
- How do we make it **scalable & convincing?**

Evaluating GNN Designs

- **Evaluating a design dimension**: controlled random search
  - **a) Sample random model-task configurations**, perturb `BatchNorm = [True, False]`
  - **Here we control the computational budget for all the models**

(a) Controlled Random Search

GNN Design Space					GNN Task Space	
BatchNorm	Activation	…	Message layers	Layer Connectivity	Task level	dataset
True	relu	…	8	skip_sum	node	CiteSeer
False	relu	…	8	skip_sum	node	CiteSeer
True	relu	…	2	skip_cat	graph	BZR
False	relu	…	2	skip_cat	graph	BZR
…						
True	prelu	…	4	stack	graph	scale free
False	prelu	…	4	stack	graph	scale free

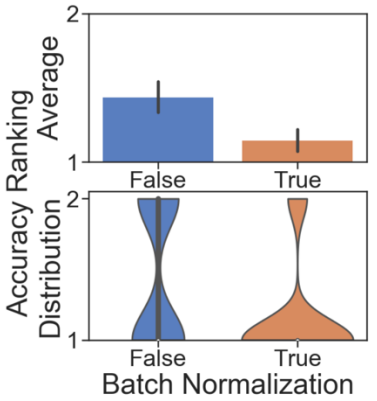
Evaluating GNN Designs

- **b) Rank** `BatchNorm = [True, False]` by their performance (lower ranking is better)
- **c) Plot Average / Distribution of the ranking** of `BatchNorm = [True, False]`

(b) Rank Design Choices by Performance

GNN Design Space	Experimental Results	
BatchNorm	Val. Accuracy	Design Choice Ranking
True	0.75	1
False	0.54	2
True	0.88	1 (a tie)
False	0.88	1 (a tie)
True	0.89	1
False	0.36	2

(c) Ranking Analysis



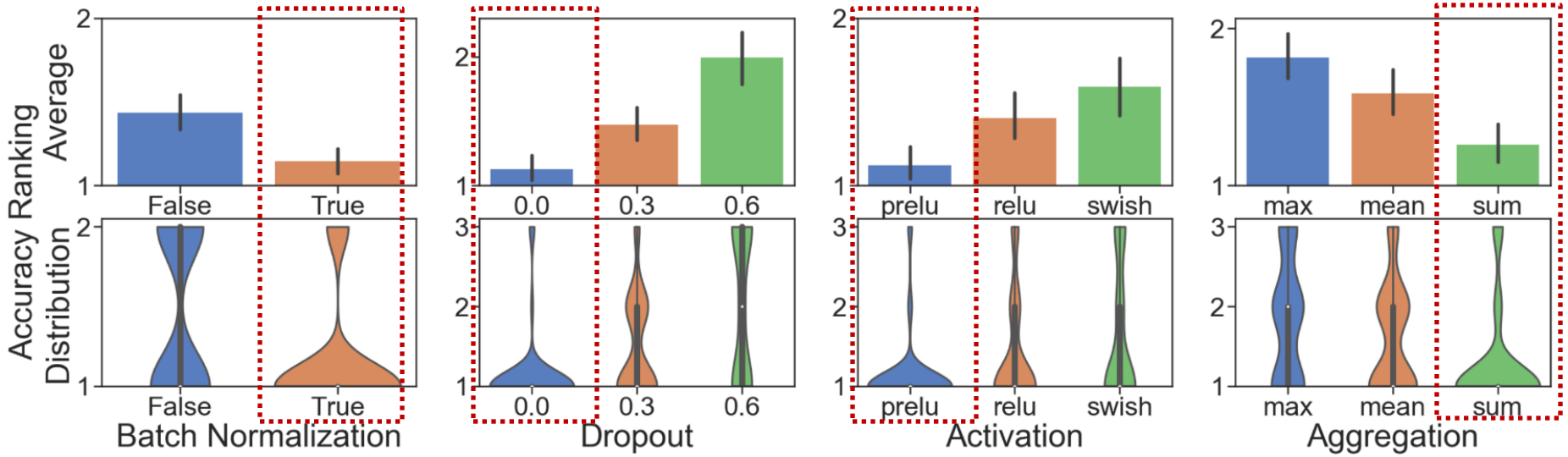
- **Summary: Convincingly evaluate any new design dimension**, e.g., evaluate a new GNN layer we propose

Results 1: A Guideline for GNN Design

- **Certain design choices exhibit clear advantages**
  - **Intra-layer designs:**

Explanation:
GNNs are hard to optimize

Explanation:
This is our new finding!



Explanation:
GNNs experience underfitting more often

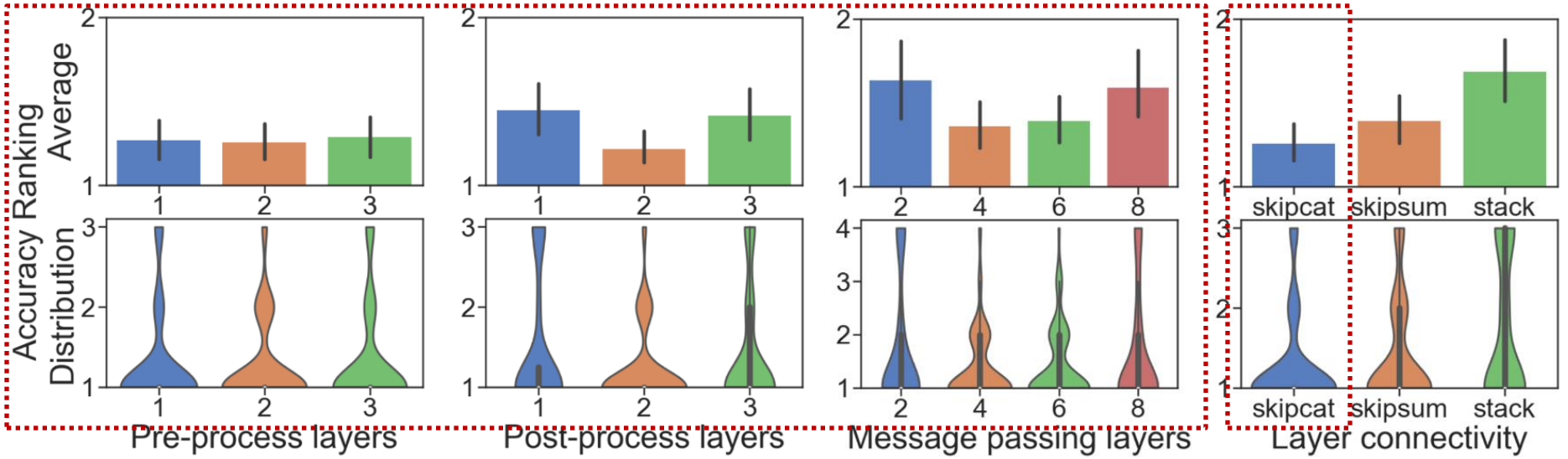
Explanation:
Sum is the most expressive aggregator

Results 1: A Guideline for GNN Design

- **Certain design choices exhibit clear advantages**
  - **Inter-layer designs**

Optimal number of layers is hard to decide

Highly dependent on the task



Explanation:

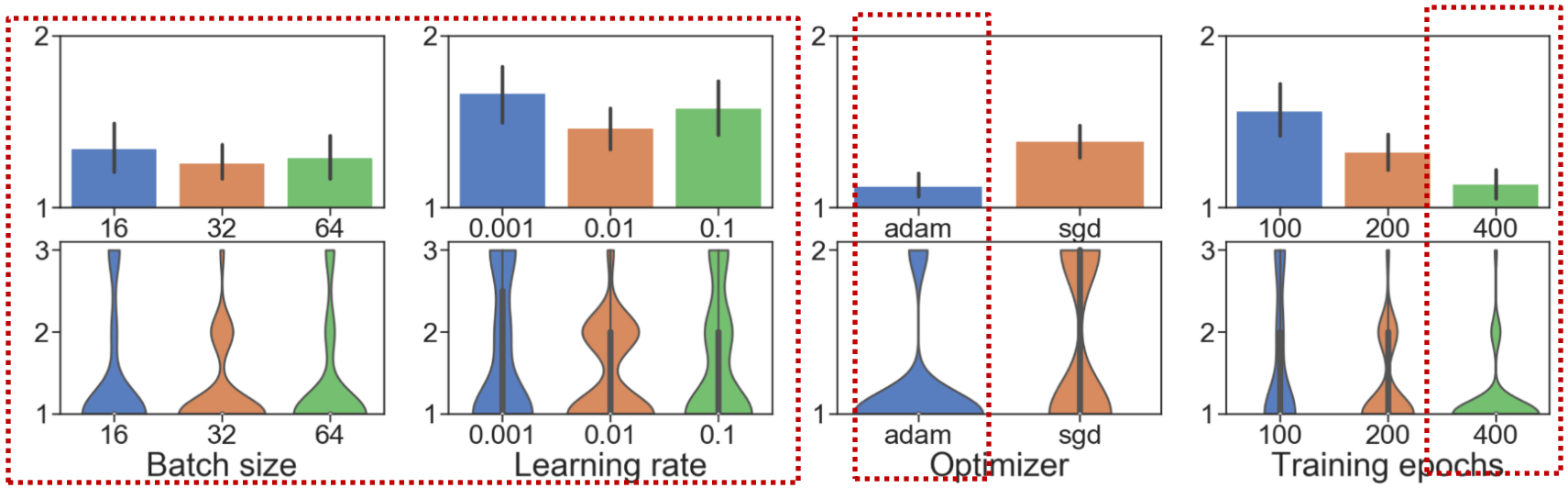
Skip connection enable hierarchical node representation

Results 1: A Guideline for GNN Design

- **Certain design choices exhibit clear advantages**
  - **Learning configurations**

Optimal batch size and learning rate is hard to decide

Highly dependent on the task



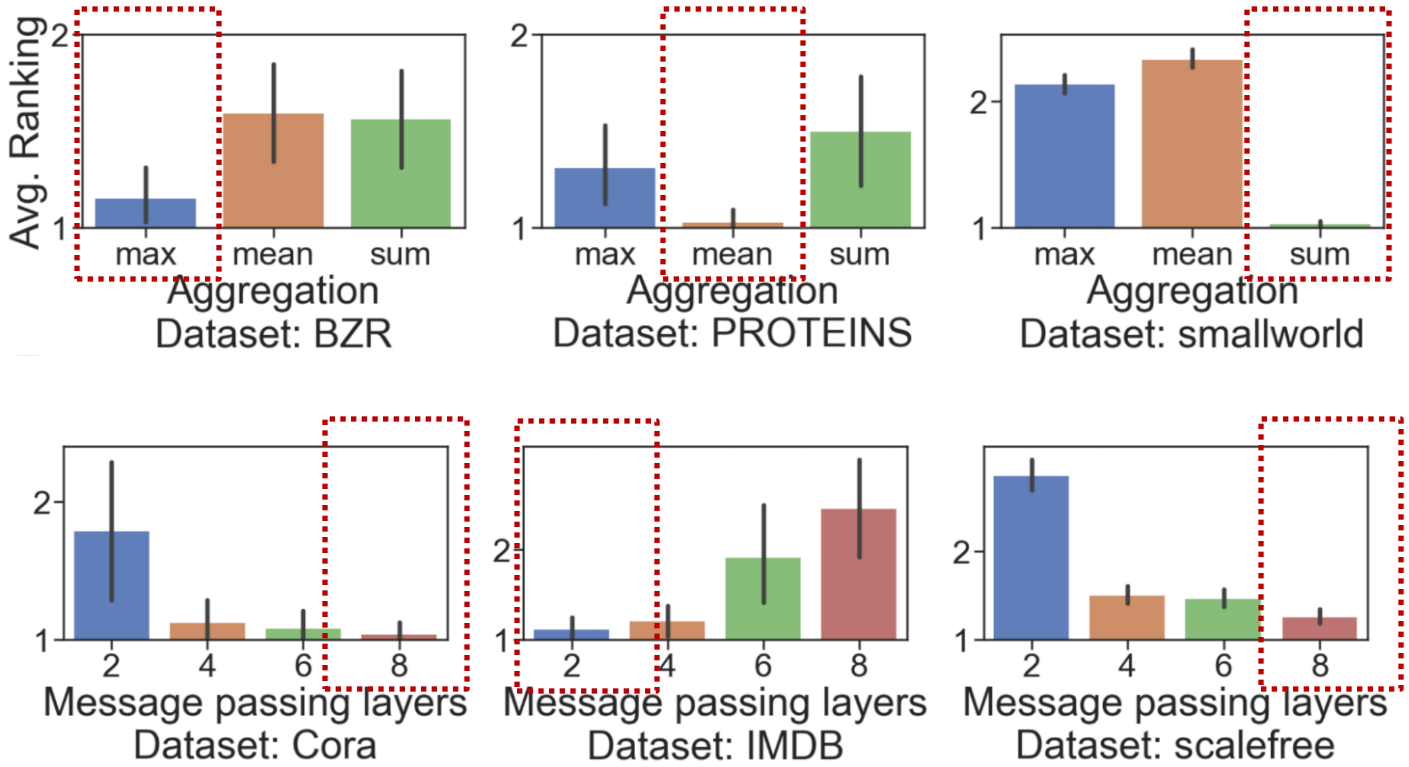
Explanation:

Adam is more robust

More training epochs is better

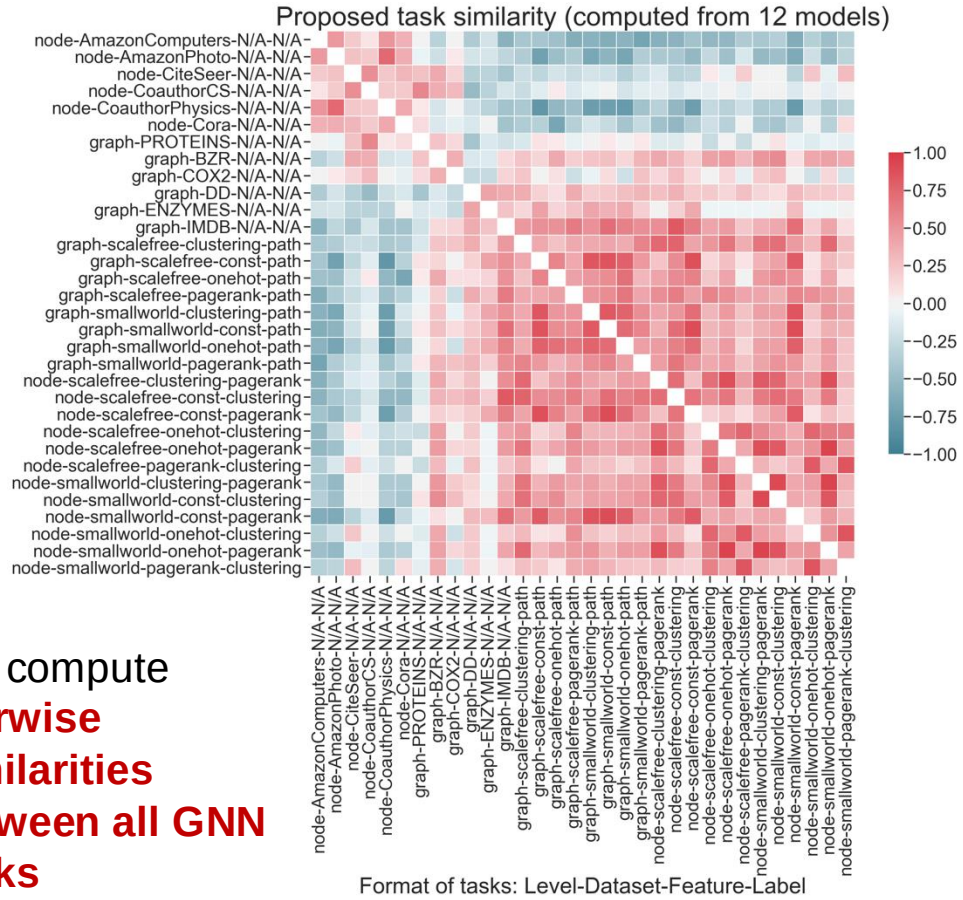
Results 2: Understanding GNN Tasks

- **Best GNN designs in different tasks vary significantly**
  - **Motivate that studying a task space is crucial**



Results 2: Understanding GNN Tasks

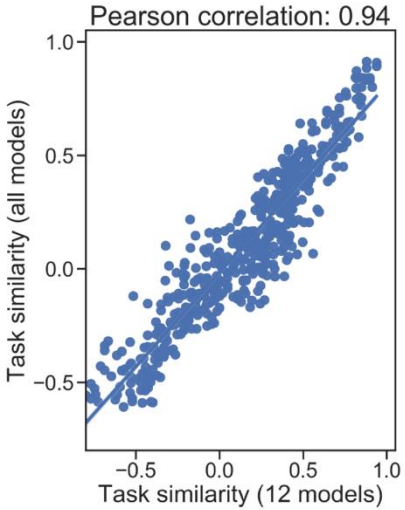
- **Build a GNN task space**



We compute **pairwise similarities between all GNN tasks**

Recall how we **compute task similarity**

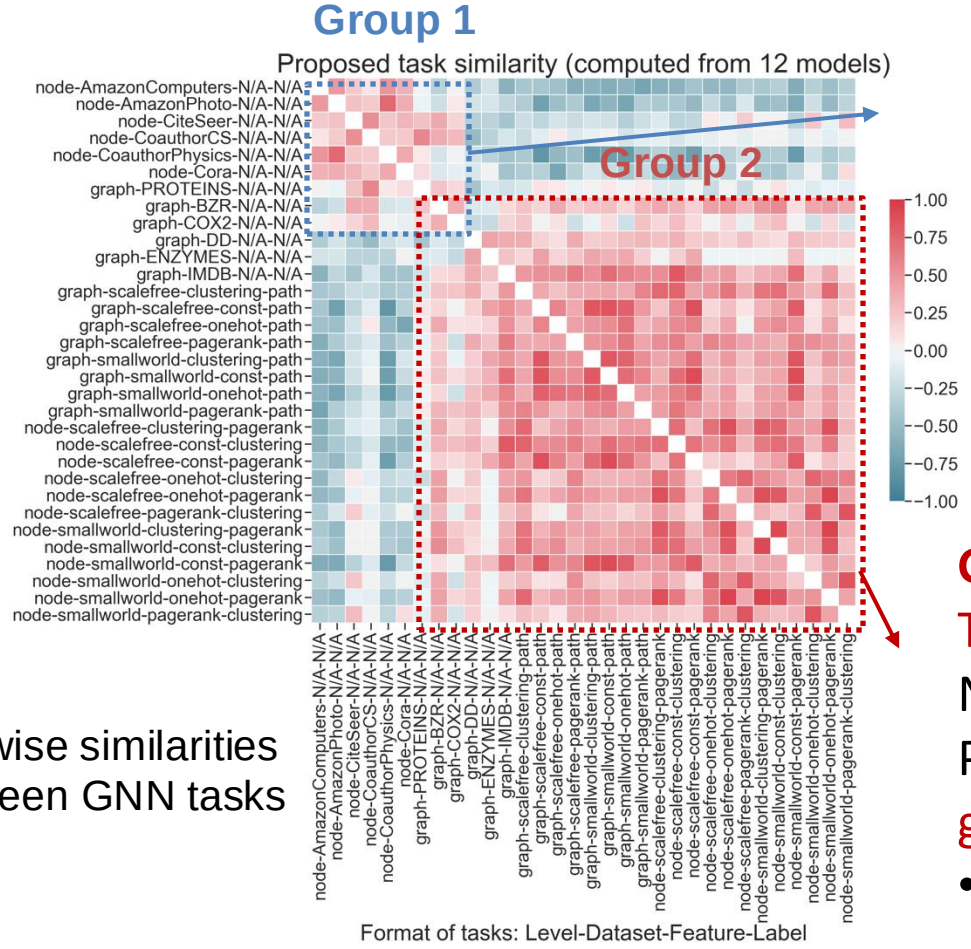
	Anchor Model Performance ranking					Similarity to Task A
Task A	M_1	M_2	M_3	M_4	M_5	1.0
Task B	M_1	M_3	M_2	M_4	M_5	0.8
Task C	M_5	M_1	M_4	M_3	M_2	-0.4



Task similarity computation is cheap: Using **12 anchor models** is a good approximation!

Results 2: Understanding GNN Tasks

- **Proposed GNN task space is informative**



Pairwise similarities between GNN tasks

Group 1:

Tasks rely on **feature information**

Node/graph classification tasks, where input graphs have high-dimensional features

- Cora graph has 1000+ dim node feature

Group 2:

Tasks rely on **structural information**

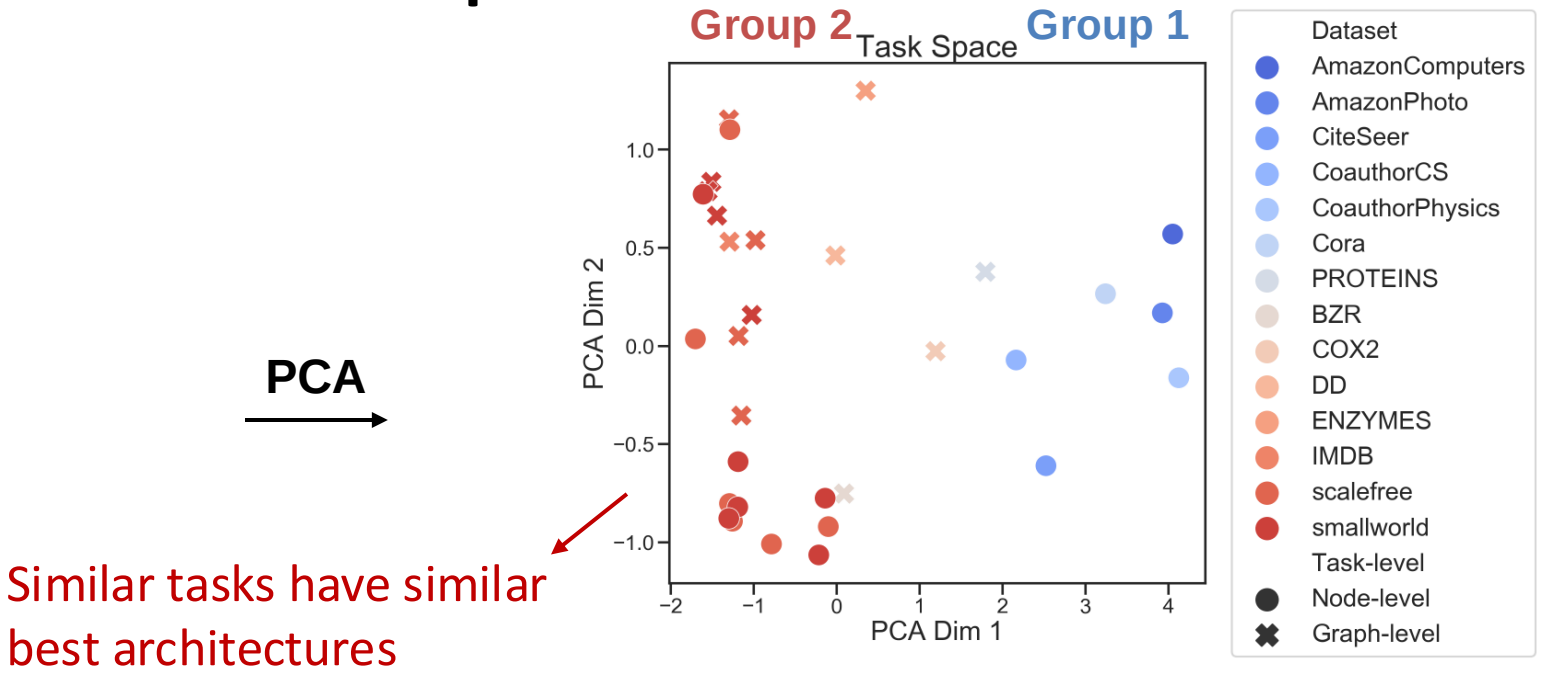
Nodes have few features

Predictions are highly dependent on graph structure

- Predicting clustering coefficients

Results 2: Understanding GNN Tasks

- **Proposed GNN task space is informative**



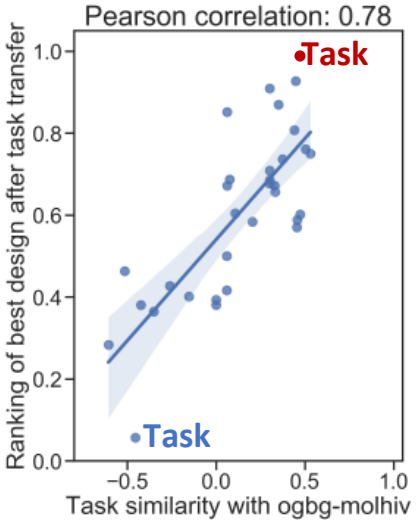
Similar tasks have similar best architectures

Best GNN Designs Found in Different Tasks					
	Pre layers	MP layers	Post layers	Connectivity	AGG
Task A	2	8	2	skip-sum	sum
Task B	1	8	2	skip-sum	sum
Task C	2	6	2	skip-cat	mean

Results 3: Transfer to Novel Tasks

- **Case study**: generalize best models to **unseen** OGB ogbg-molhiv task
  - **ogbg-molhiv is unique from other tasks**: 20x larger, imbalanced (1.4% positive) and requires out-of-distribution generalization

- **Concrete steps for applying to a novel task:**
  - **Step 1:** Measure 12 anchor model performance on the new task
  - **Step 2:** Compute similarity between the new task and existing tasks
  - **Step 3:** Recommend the best designs from existing tasks with high similarity



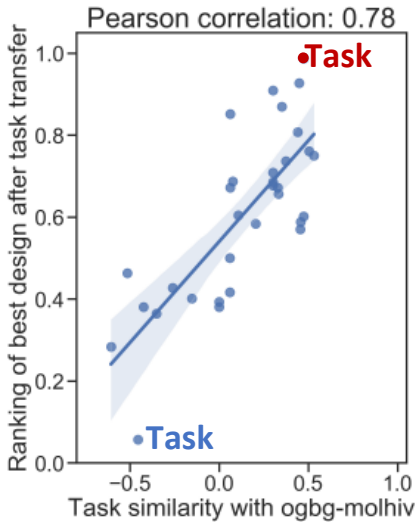
Results 3: Transfer to Novel Tasks

- **Our task space can guide best model transfer to novel tasks!**

We pick 2 tasks:

Task A: Similar to OGB

Task B: Not similar to OGB



Findings:

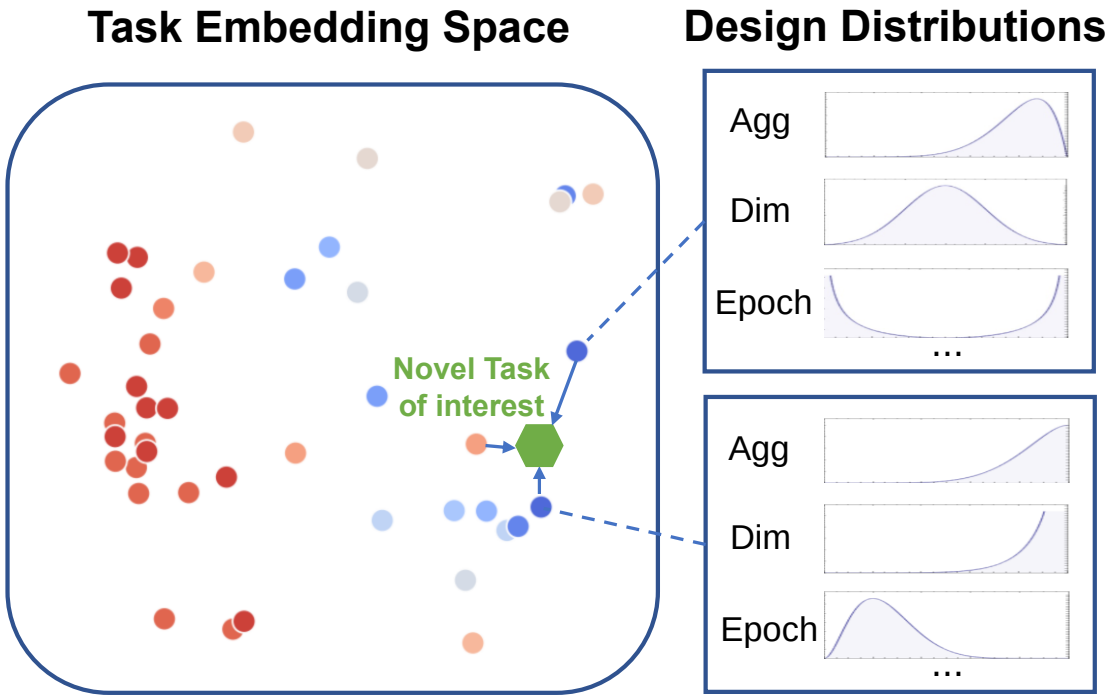
Transfer the best model from Task A achieves SOTA on OGB

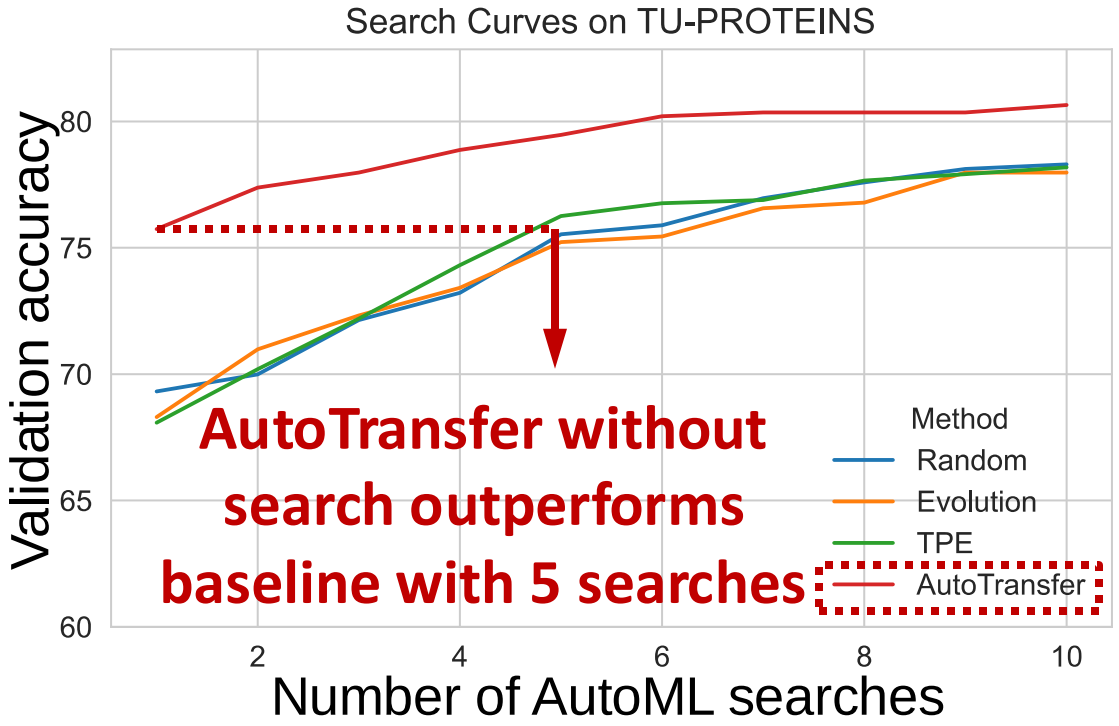
Transfer the best model from Task B performs badly on OGB

	Task A: graph-scalefree-const-path	Task B: node-CoauthorPhysics
Best design in our design space	(1, 8, 3, skipcat, sum)	(1, 4, 2, skipcat, max)
Task Similarity with ogbg-molhiv	**0.47**	**-0.61**
Performance after transfer to ogbg-molhiv	***0.785***	***0.736***

Previous SOTA: 0.771

Extend to **General AutoML Problems**



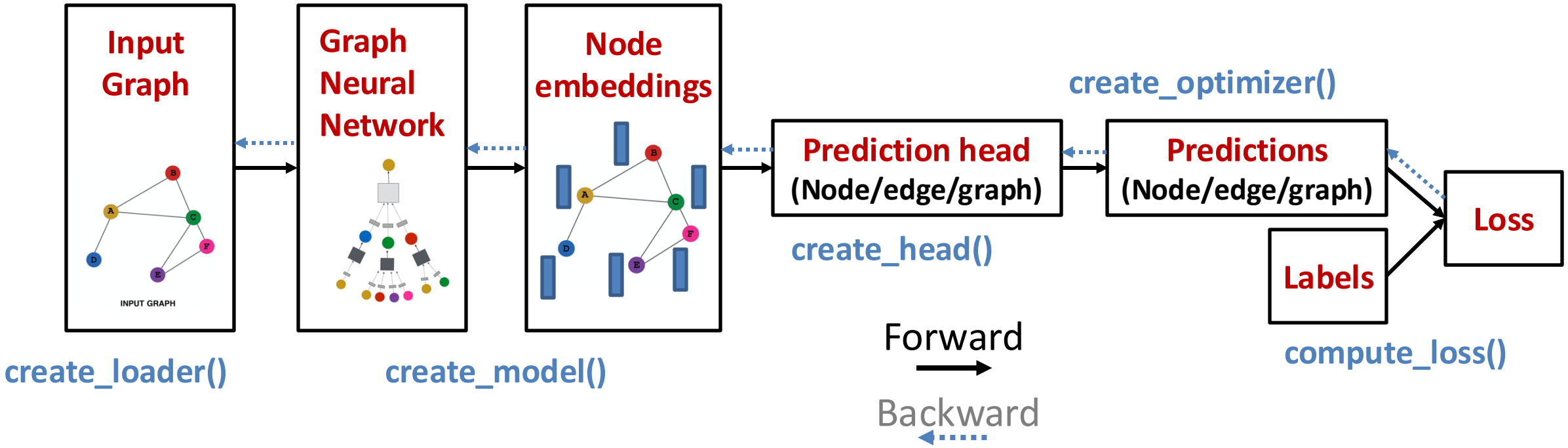


Sample from the design distributions of the most **similar tasks** through the **quantitate task similarity metric**

Given a new task, AutoTransfer significantly **saves the AutoML search cost**

Software for Graph ML- **GraphGym**

- **GraphGym: powerful & easy-to-use primitives** for **graph ML**
  - Users can **quickly find the best GNN design** for **their specific tasks**



GraphGym: Easy-to-use API

GraphGym config

```yaml
out_dir: results
dataset:
  format: PyG
  name: Cora
  task: node
  task_type: classification
  node_encoder: False
  node_encoder_name: Atom
  edge_encoder: False
  edge_encoder_name: Bond
train:
  batch_size: 128
  eval_period: 1
  ckpt_period: 100
  sampler: full_batch
model:
  type: gnn
  loss_fun: cross_entropy
  edge_decoding: dot
  graph_pooling: add
gnn:
  layers_pre_mp: 0
  layers_mp: 2
  layers_post_mp: 1
  dim_inner: 16
  layer_type: gcnconv
  stage_type: stack
  batchnorm: False
  act: prelu
  dropout: 0.1
  agg: mean
  normalize_adj: False
optim:
  optimizer: adam
  base_lr: 0.01
  max_epoch: 200
```

→ **Dataset**

→ **Training**

→ **Model**

→ **Optimizer**

- **GraphGym:** A config **fully describes an end-to-end GNN experiment**
  - **Run an experiment in 1 line**

```
python main.py --cfg example_node.yaml --repeat 3
```

  - **Run batch of experiments in 2 lines**

```
# generate configs
python configs_gen.py --config example.yaml --grid example.txt
# launch batch of experiments
bash run_batch.sh configs/example_grid_example 3 8
```

  - **Democratize Graph ML** for domain experts and practitioners

GraphGym Accelerates GNN Research

Proposed a **theoretically expressive graph ML model: ID-GNN**

Experimentally show ID-GNN is better than **GNN**

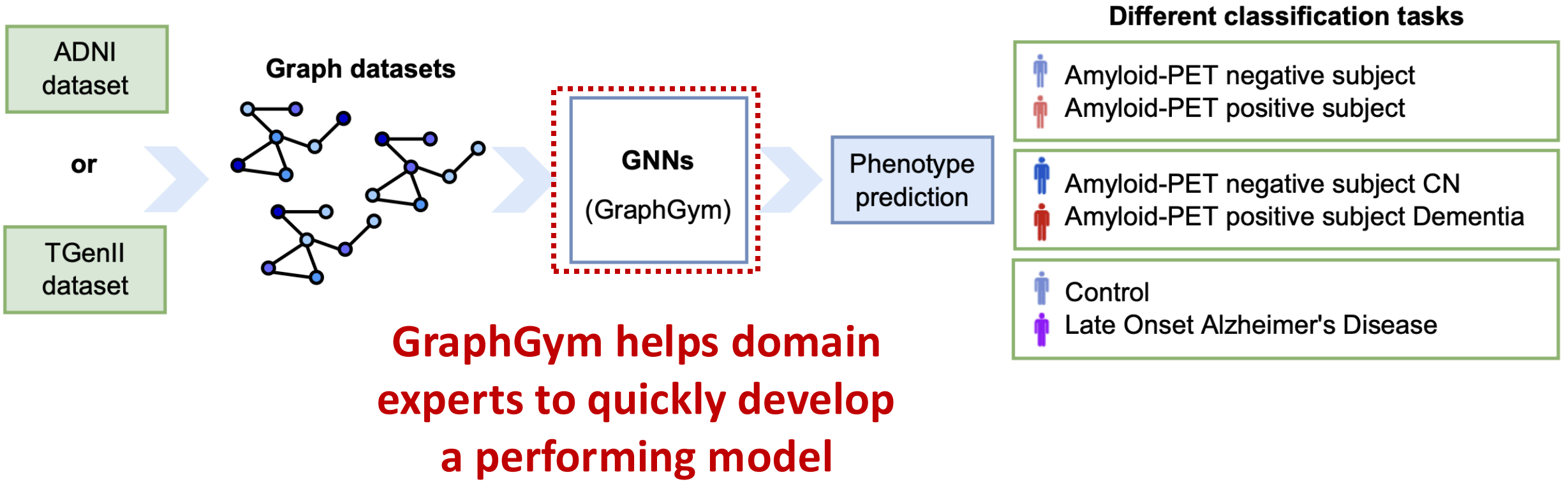
2 types of layers ✖ **4 variants** of models ✖ **20 tasks** node/edge/graph ✖ **3 random seeds** = **480 Experiments**

Computational budget controlled

- **2 lines of code with GraphGym**
  - **Accelerate research** and **promote knowledge sharing** (reproducibility)

```
# generate configs
python configs_gen.py --config baseline.yaml --config_budget baseline.yaml --grid idgnn.txt
# run batch of configs
# Args: config_dir, num of repeats, max jobs running, sleep time
bash run_batch.sh configs/baseline_grid_idgnn 3 10 1
```

GraphGym **Accelerates Scientific Discovery**

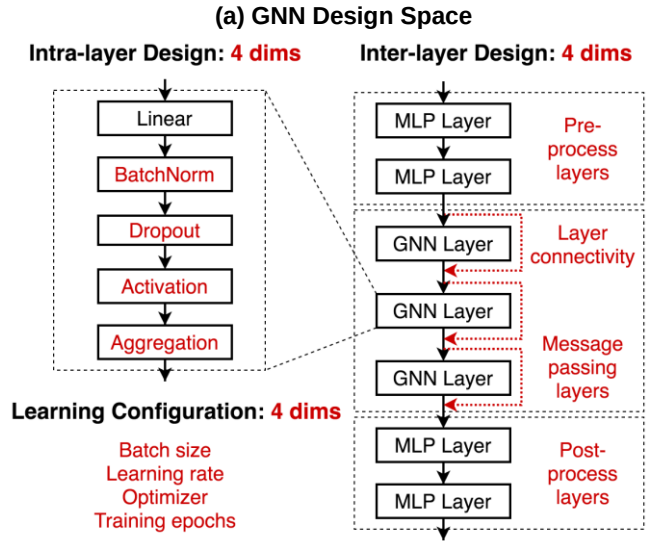


GraphGym helps domain experts to quickly develop a performing model

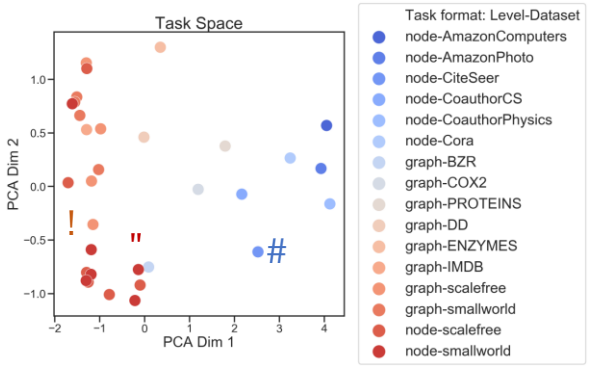
Hernández-Lorenzo, et al. "Graph Neural Networks for the Early Diagnosis of Alzheimer's Disease." ***Nature Scientific Reports, 2022***

GNN Design Space: Summary

- The first systematic investigation of:
  - **General guidelines for GNN design**
  - **Understandings of GNN tasks**
  - **Transferring best GNN designs across tasks**
  - **GraphGym:** Easy-to-use **code platform for GNN**



(b) GNN Task Space



Best GNN Designs Found in Different Tasks					
	Pre layers	MP layers	Post layers	Connectivity	AGG
Task !	2	8	2	skip-sum	sum
Task "	1	8	2	skip-sum	sum
Task #	2	6	2	skip-cat	mean

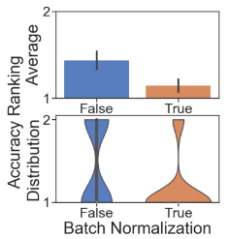
(c) Controlled Random Search

GNN Design Space					GNN Task Space	
BatchNorm	Act	…	MP layers	Connectivity	level	dataset
True	relu	…	8	skip_sum	node	CiteSeer
False	relu	…	8	skip_sum	node	CiteSeer
True	relu	…	2	skip_cat	graph	BZR
False	relu	…	2	skip_cat	graph	BZR
…						

(d) Rank Design Choices by Performance

Experimental Results	
Val. Accuracy	Design Choice Ranking
0.75	1
0.54	2
0.88	1 (a tie)
0.86	1 (a tie)
…	

(e) Ranking Analysis



GraphGym → PyG

- GraphGym is integrated to PyG
- **PyG**: now **the most popular graph learning library**
  - **21,000+ Github Stars, 130,000+ monthly downloads**
  - The **go-to library for GNN researchers**: used in **1600 research papers**
  - **Widely used in industry:** Airbus, Amazon, Aramco, AstraZeneca, ...

```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
```

GNN Implementation

PyTorch Geometric: Practical Realization of Graph Neural Networks

Implementing Graph Neural Networks is challenging

- Sparsity *and* irregularity of the underlying data
  - How can we effectively parallelize irregular data of potentially varying size?
- Heterogeneity of the underlying data
  - numerical, categorical, image and text features, potentially over *different* types of data
- *Inherently* dynamic
  - it is hard to find scenarios in which graphs will *not* change over time
- Various *different* requests on scalability
  - sparse *vs.* dense graphs, many small *vs.* single giant graphs, ...
- Applicability to a set of *diverse* tasks
  - node-level *vs.* link-level *vs.* graph-level, clustering, pre-training, self-supervision, ...

PyTorch Geometric

- **PyG** *(PyTorch Geometric)* is the PyTorch library to unify deep learning on graph-structured data.
  - simplifies implementing and working with Graph Neural Networks
  - bundles state-of-the-art GNN architectures and training procedures
  - achieves high GPU throughput on **sparse data** of varying size
  - suited for both academia and industry
    - flexible, comprehensive, easy-to-use

Design Principles

- **Graph-based Neural Network Building Blocks**
  - Message Passing layers
  - Normalization layers
  - Pooling & Readout layers
- **Graph Transformations & Augmentations**
  - Graph diffusion
  - Missing feature value imputation
  - Mesh and Point Cloud support

Design Principles

- **In-Memory Graph Storage, Datasets & Loaders**
  - Support for heterogeneous graphs 200+ benchmark datasets
  - 10+ sampling techniques
- **Examples & Tutorials**
  - Learn practically about GNNs
  - Videos, Colabs & Blogs
  - Application-driven Graph ML Tutorials
  - Notebooks examples with visualization

Design Principles

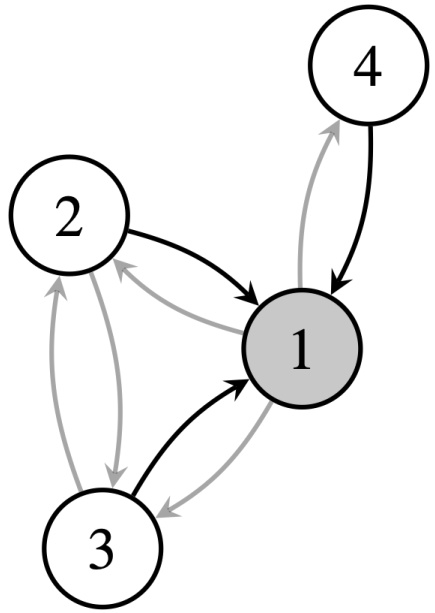
- PyG is highly modular
  - Access to 200+ datasets and 50+ transforms
  - Access to a variety of mini-batch loaders
    - Node-wise sampling, subgraph-wise sampling, graph-wise batching
  - Access to 80+ GNN layers, normalizations and readouts as neural network building blocks 20+ pre-defined models
    - SAGEConv, GCNConv, GATConv, GINConv, PNAConv, …
    - GraphSAGE, GCN, GAT, GIN, PNA, SchNet, DimeNet, ...
  - Access to regular PyTorch loss functions and training routines
    - Classification, Regression, Self-Supervision, ... Node-level, Link-level, Graph-level

Design Principles

```
dataset = Reddit(root_dir, transform)

loader = NeighborLoader(dataset, num_neighbors=[25, 10])

class GNN(torch.nn.Module):
    def __init__(self):
        self.conv1 = SAGEConv(F_in, F_hidden)
        self.conv2 = SAGEConv(F_hidden, F_out)

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

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

Implementing Graph Neural Networks

- Given a sparse graph $\mathcal{G} = \left(\boldsymbol{H}^{(0)}, (\boldsymbol{I}, \boldsymbol{E})\right)$ with
  - Input node features $\boldsymbol{H}^{(0)} \in \mathbb{R}^{|\mathcal{V}| \times C}$
  - Edge indices $\boldsymbol{I} \in \{1, 2, \ldots, |\mathcal{V}|\}^{2 \times |\mathcal{E}|}$
  - Optional edge features $\boldsymbol{E} \in \mathbb{R}^{|\mathcal{E}| \times D}$
- **Message Passing Scheme**
  - permutation-invariant aggregation operator, e.g., sum, mean or max
  - $\boldsymbol{h}_i^{(\ell+1)} = \text{Update}_\theta(\boldsymbol{h}_i^{(\ell)}, \bigoplus_{j \in \mathcal{N}(i)} \text{Message}_\theta(\boldsymbol{h}_i^{(\ell)}, \boldsymbol{h}_i^{(\ell)}, \boldsymbol{e}_{i,j}))$



Graph Creation

- **PyG** expects a graph in either COO or CSR format to model complex (heterogeneous graphs)
- Both nodes and edges can hold any set of curated features

```
data = HeteroData()
data['user'].x = ...
data['product'].x = ...
data['user', 'product'].edge_index = ...
data['user', 'product'].edge_attr = ...
data['user', 'product'].time = ...
```

- Benchmarking graphs: Training labels and dataset splits are pre-defined
- Real-world graphs in applications: Build/Bring your own training labels and dataset splits

Best practices for graph design suitable for message passing

Best practices for feature selection

Best practices for feature encoding suitable for neural networks

Task Formulation

- **PyG** supports any graph-related machine learning task, but curation of training labels is a user task

```
# Add user-level node labels:
data['user'].y = ...

transform = RandomNodeSplit(num_val=0.1, num_test=0.1)
data = transform(data)
```

- Benchmarking graphs: Training labels and dataset splits are pre-defined
- Real-world graphs in applications: Build/Bring your own training labels and dataset splits

Best practices for curating training labels to map to ML task

Best practices for avoiding data leakage in temporal scenarios

Best practices for dataset splitting to match real-world use-cases

GNN Implementation

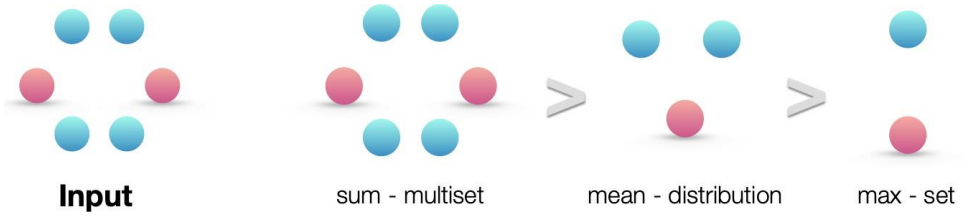
PyTorch Geometric: Advanced Features

PyG Highlights

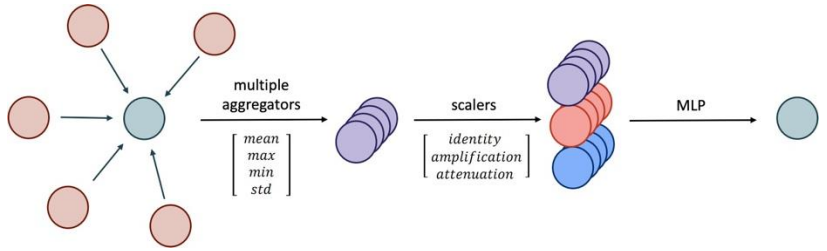
- Advanced Features
  - **Improved GNN design via principled aggregations**
  - Improved efficiency on heterogeneous graphs
  - Out-of-core data interfaces
  - Explainability and Compilation
  - GNN + LLM

Principled Aggregations

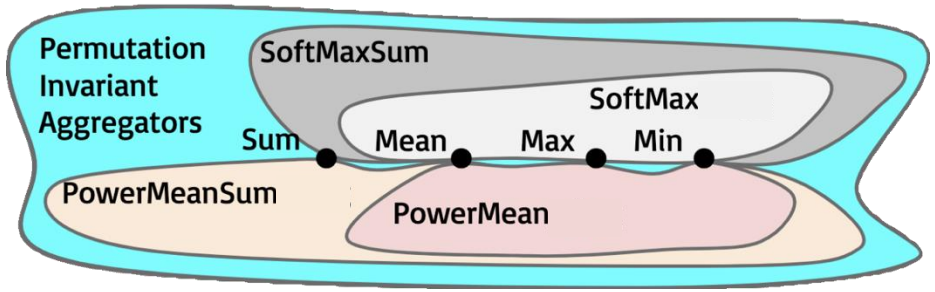
- Choice of neighborhood aggregation is a central topic in Graph ML research



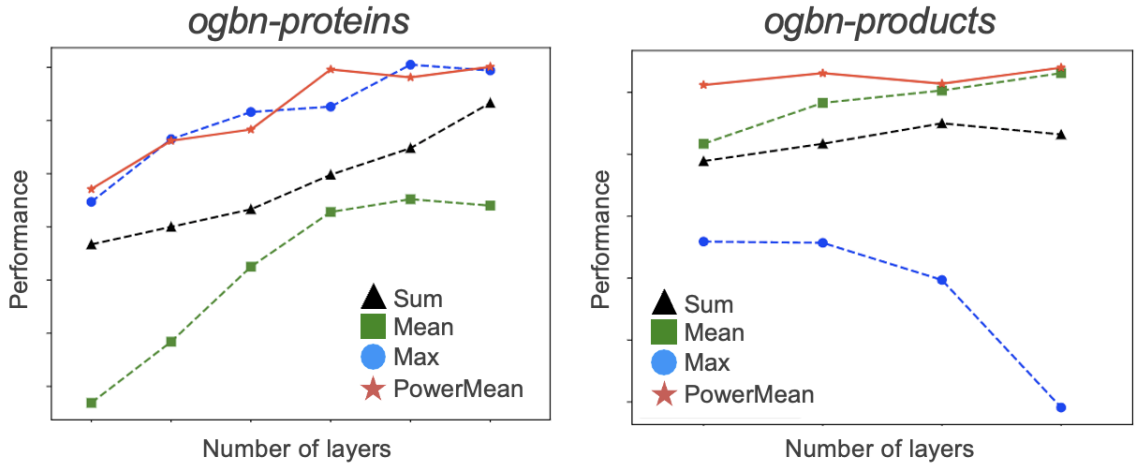
Xu *et al.*: How Powerful Are Graph Neural Networks?



Corso *et al.*: Principal Neighborhood Aggregation for Graph Nets

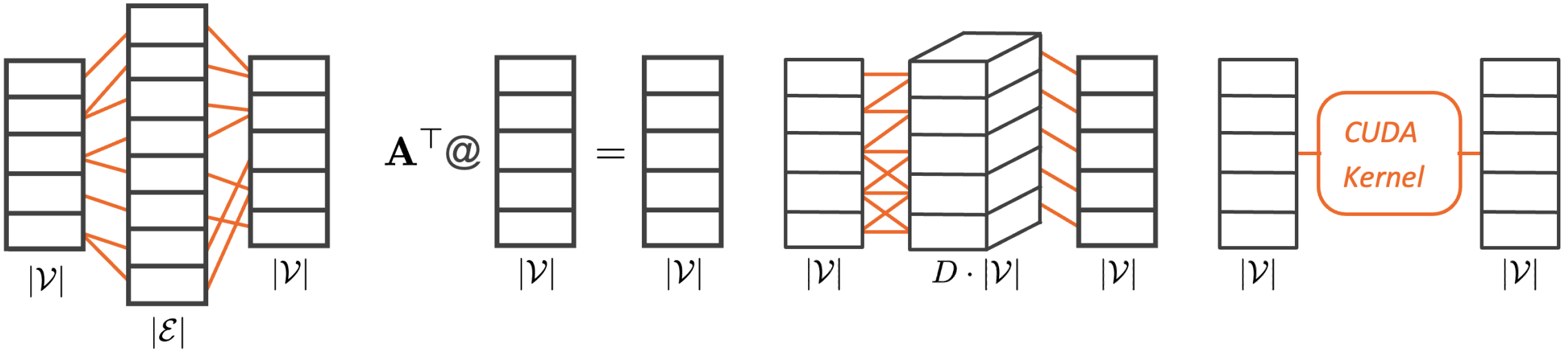


Li *et al.*: Deeper-GCN: All You Need to Train Deeper GCNs



Principled Aggregations

The *different* flavors of implementing aggregations



Gather & Scatter (Py>=0.1)

- very flexible 🤗🤗
- fast for sparse graphs 🤗🤗
- memory-inefficient 😭😭

Sparse MatMul (Py>=1.6)

- less flexible 😭😭
- very fast 🤗🤗
- memory-efficient 🤗🤗

Degree Bucketing (Py>=2.1)

- very flexible 🤗🤗
- fast for sparse graphs 🤗🤗
- memory-inefficient 😭😭

Individual Kernel (Py>=2.2)

- not flexible at all 😭😭
- memory-efficient 🤗🤗
- very fast 🤗🤗

Principled Aggregations

- PyG makes the concept of aggregations a first-class principle
  - Access to all kinds of simple, advanced, learnable and exotic aggregations
    - Median, Softmax, Attention, LSTM, ...
  - Fully-customize and combine aggregations within *MessagePassing* or for global pooling
  - Aggregations will pick up the best format to accelerate computation
    - scatter reductions, degree bucketing, ...
  - Further optimization via fusion
    - minimize I/O from global memory

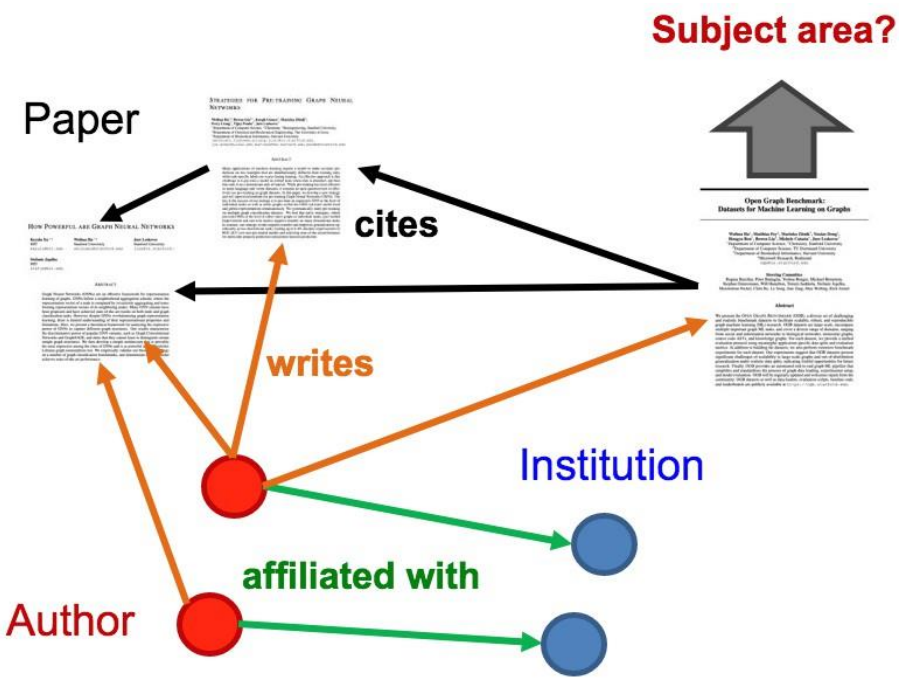
Principled Aggregations

```python
# Simple aggregations:
mean_aggr = aggr.MeanAggregation()
max_aggr = aggr.MaxAggregation()

# Advanced aggregations:
median_aggr = aggr.MedianAggregation()

# Learnable aggregations:
softmax_aggr = aggr.SoftmaxAggregation(learn=True)
powermean_aggr = aggr.PowerMeanAggregation(learn=True)

# Exotic aggregations:
lstm_aggr = aggr.LSTMAggregation()
sort_aggr = aggr.SortAggregation(k=4)

# Use within message passing:
conv = MyConv(aggr=[median_agr, lstm_aggr])

# Use for global pooling:
h_graph = sort_aggr(h_node, batch)
```

PyG Highlights

- Advanced Features
  - Improved GNN design via principled aggregations
  - **Improved efficiency on heterogeneous graphs**
  - Out-of-core data interfaces
  - Explainability and Compilation
  - GNN + LLM

Heterogeneous Graph Support



- (Nearly) all real-world graphs are heterogeneous!

Heterogeneous Graph Support

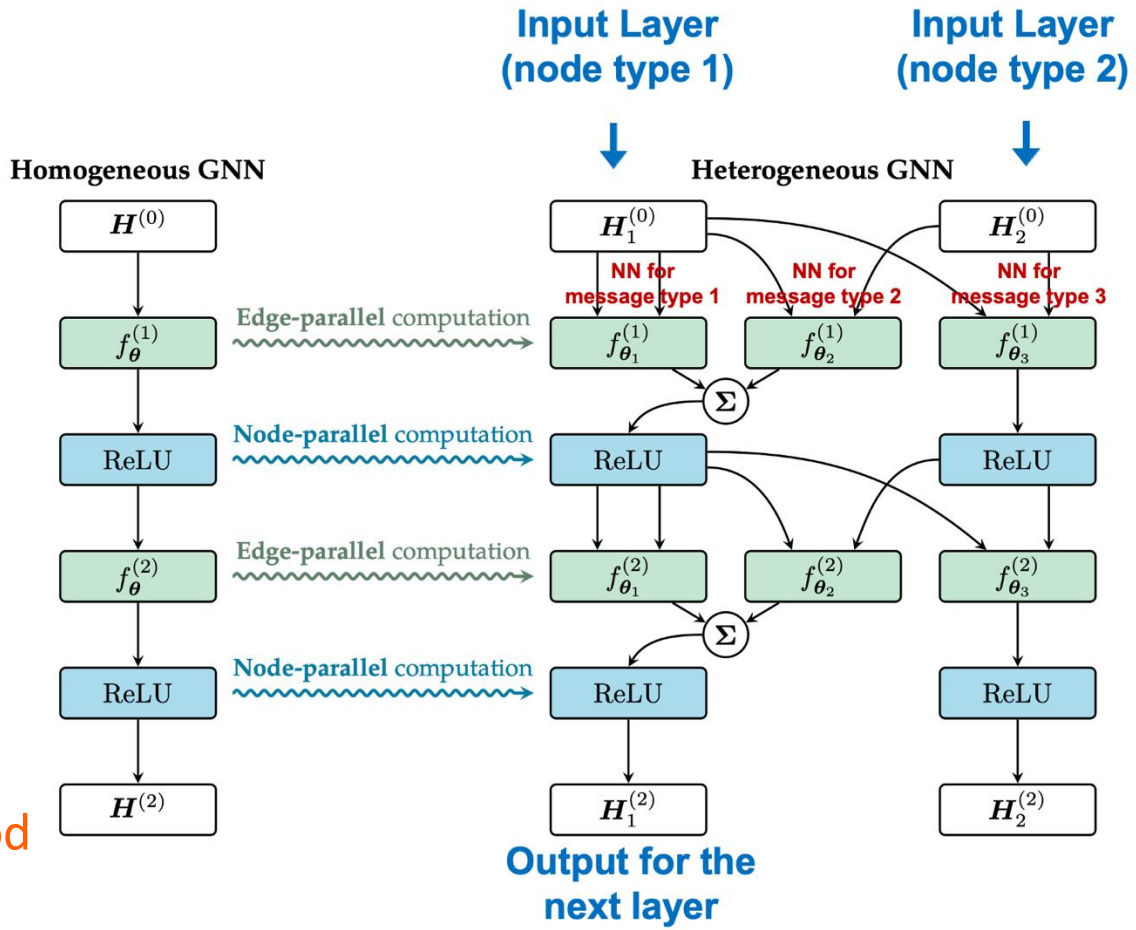
- Heterogeneous Graph Neural Networks
  - A homogeneous GNN can be converted to a heterogeneous one by learning distinct parameters for each individual edge type

Edge type dependent parameters

$$\boldsymbol{h}_i^{(\ell+1)} = \sum_{r \in \mathcal{R}} f_{\theta_r}^{(\ell+1)}(\boldsymbol{h}_i^{(\ell)}, \{\boldsymbol{h}_j^{(\ell)} : j \in \mathcal{N}^{(r)}(i)\})$$

The number of relations

Relational-wise neighborhood



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)):

Implemented via torch.fx

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)
```

PyG Highlights

- Advanced Features
  - Improved GNN design via principled aggregations
  - Improved efficiency on heterogeneous graphs
  - **Out-of-core data interfaces**
  - Explainability and Compilation
  - GNN + LLM

Out-of-core Data Interfaces

There exists multiple ways to scale PyG beyond single-node in-memory datasets

1. Pre-process subgraphs on disk and load them on-the-fly,
e.g., via dedicated Spark routines

- Naturally supported via PyG's mini-batch DataLoader
- No sampling/feature fetching overhead during GNN training
- Heavy pre-processing and storage requirements
- Experimentation with different sampling strategies and parameters gets increasingly harder

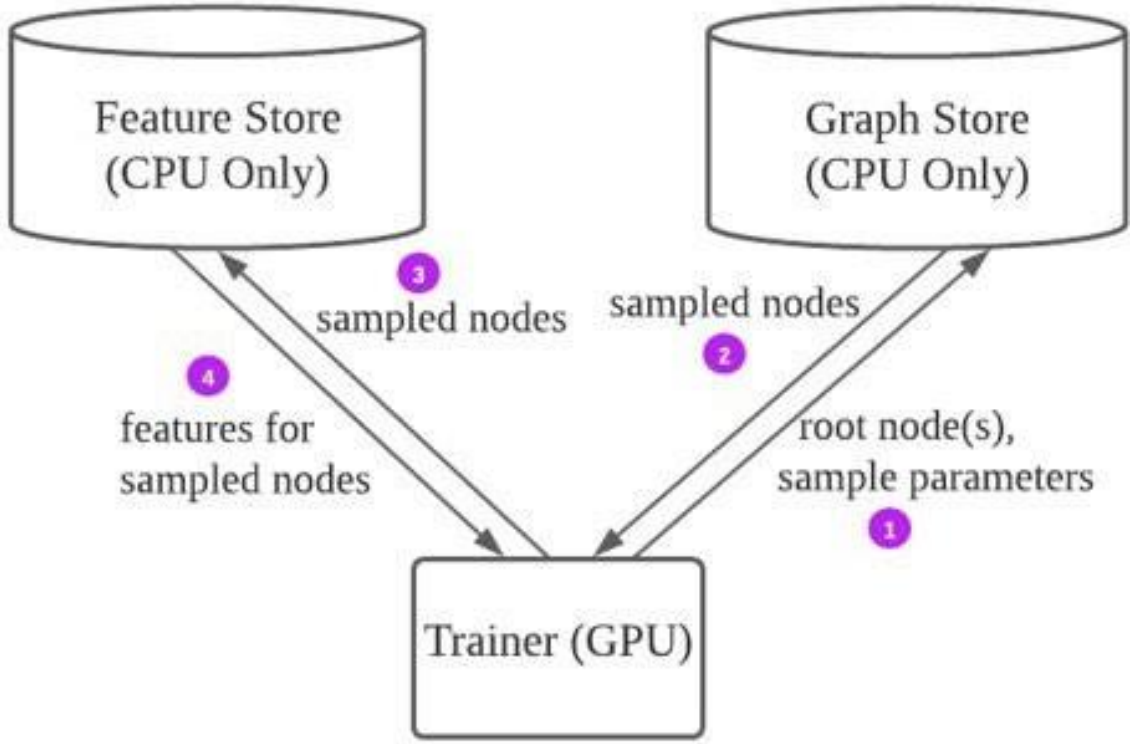
```
class Dataset:
    def __getitem__(self, idx) → Data:
        # 1. Load subgraph from disk:
        data = load()
        # 2. Convert to PyG format:
        data = Data()
        return data

dataset = Dataset()
loader = DataLoader(dataset, batch_size=128)
```

Out-of-core Data Interfaces

There exists multiple ways to scale PyG beyond single-node in-memory datasets

2. With PyG, we support any backend by providing FeatureStore and GraphStore abstractions

- Disentangles feature fetching from graph sampling routines
- Allows for distributed server/client architectures
- Allows for out-of-memory backends, e.g., via connecting to graph databases



Out-of-core Data Interfaces

There exists multiple ways to scale PyG beyond single-node in-memory datasets

2. With PyG, we support any backend by providing FeatureStore and GraphStore abstractions

- These interfaces can enable an elastic, heterogeneous compute architecture
  - Read-optimized on-disk feature store via RocksDB
  - Low-latency in-memory graph store

We are working with multiple graph database vendors to enable store implementations, e.g., KùzuDB

```python
class MyFeatureStore(FeatureStore):
    def get_tensor(self, attr):
        pass  # Implement feature access


class MyGraphStore(GraphStore):
    def sample_from_nodes(self, index):
        pass  # Implement node-wise sampling

    def sample_from_edges(self, index):
        pass  # Implement edge-wise sampling
```

PyG Highlights

- Advanced Features
  - Improved GNN design via principled aggregations
  - Improved efficiency on heterogeneous graphs
  - Out-of-core data interfaces
  - **Explainability and Compilation**
  - GNN + LLM

Explainability

- PyG 2.3 introduced a new unified explainability interface to explain any GNN out-of-the-box across various explainer algorithms
  - Works on both homogeneous and heterogeneous graphs
  - Support for general explainers such as integrated gradients, saliency, shapley, etc
  - Support for dedicated GNN explainers, e.g., GNNExplainer, PGExplainer, etc
  - Support for a diverse range of tasks
  - Support for visualizations and metric computation

```
explainer = Explainer(
    model=model,
    algorithm=GNNExplainer(epochs=200),
    node_mask_type='attributes',
    edge_mask_type='object',
    model_config=dict(
        mode='multiclass_classification',
        task_level='node',
        return_type='probs',
    ),
)

explanation = explainer(x, edge_index)
```

Compilation

- PyG 2.3 takes full advantage of PyTorch 2.0 compilation, which makes your GNN run faster by JIT-compiling it into optimized kernels

Model	Mode	Forward	Backward	Total	Speedup
`GCN`	Eager	2.6396s	2.1697s	4.8093s	
`GCN`	**Compiled**	**1.1082s**	**0.5896s**	**1.6978s**	**2.83x**
`GraphSAGE`	Eager	1.6023s	1.6428s	3.2451s	
`GraphSAGE`	**Compiled**	**0.7033s**	**0.7465s**	**1.4498s**	**2.24x**
`GIN`	Eager	1.6701s	1.6990s	3.3690s	
`GIN`	**Compiled**	**0.7320s**	**0.7407s**	**1.4727s**	**2.29x**

```
model = GraphSAGE()
model = torch.compile(model)

out = model(data.x, data.edge_index)
```

PyG Highlights

- Advanced Features
  - Improved GNN design via principled aggregations
  - Improved efficiency on heterogeneous graphs
  - Out-of-core data interfaces
  - Explainability and Compilation
  - **GNN + LLM**

PyG 2.6: Combining GNNs with LLMs

- In order to facilitate further research on combining GNNs with LLMs, **PyG 2.6** introduces:
  - A new sub-package torch_geometric.nn.nlp with fast access to SentenceTransformer models and LLMs
  - A new model GRetriever that is able to co-train LLAMA2 with GAT for answering questions based on knowledge graph information
  - A new example folder examples/LLM that shows how to utilize these models in practice

Summary for Today

- **GraphGym:** Easy-to-use **code platform for GNN**
  - **General guidelines for GNN design**
  - **Understandings of GNN tasks**
  - **Transferring best GNN designs across tasks**
- **PyG principles & highlights**