# GraphGym: Easy-to-use API for Graph Learning

Collaboration with Rex Ying, Matthias Fey, Jure Leskovec


Jiaxuan You

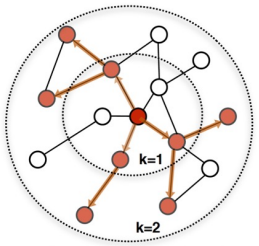
Stanford University

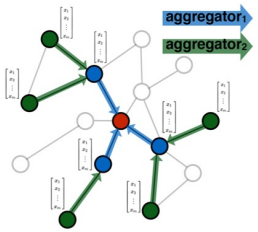
# Overview of This Talk

Graph Learning Basics

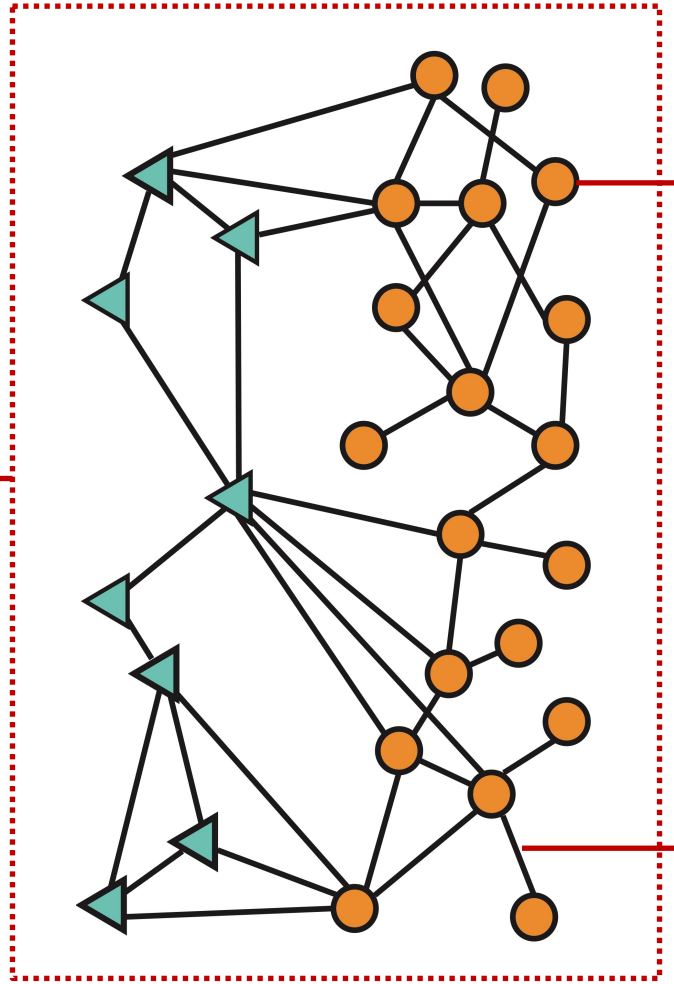
Challenges in Applying Graph Learning

**GraphGym**



# Machine Learning Tasks on Graphs



**Node-level prediction**

*"What is the area of this research paper?"*

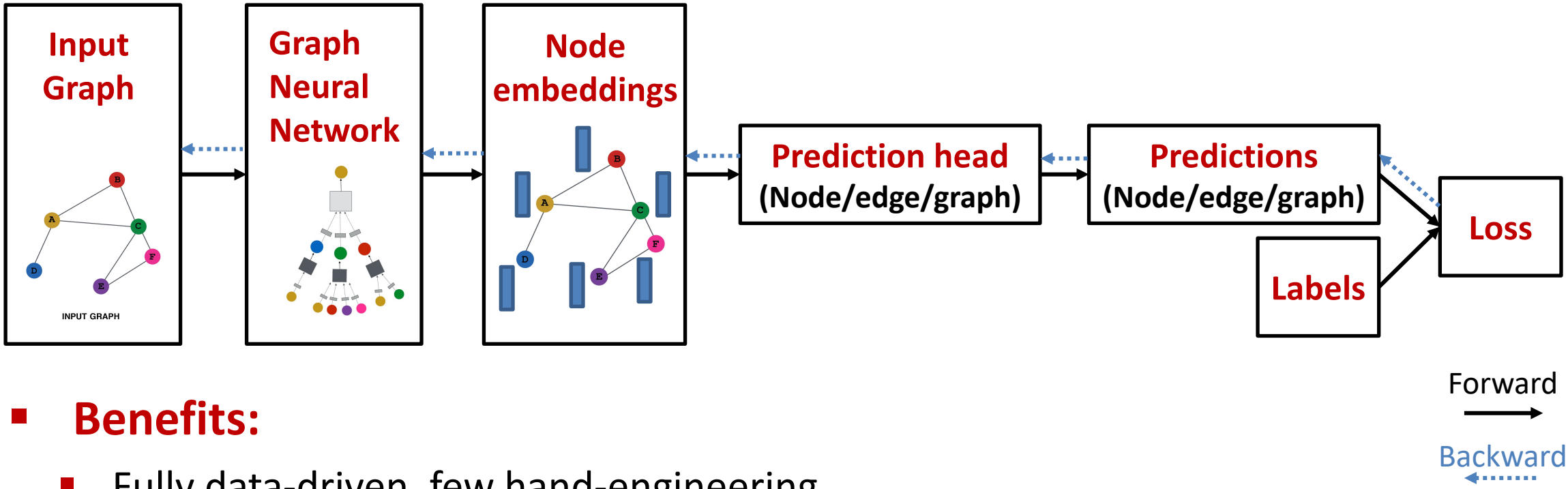
**Graph-level prediction**

*"Is this molecular graph toxic?"*

**Edge-level prediction**

*"Is this transaction fraudulent?"*

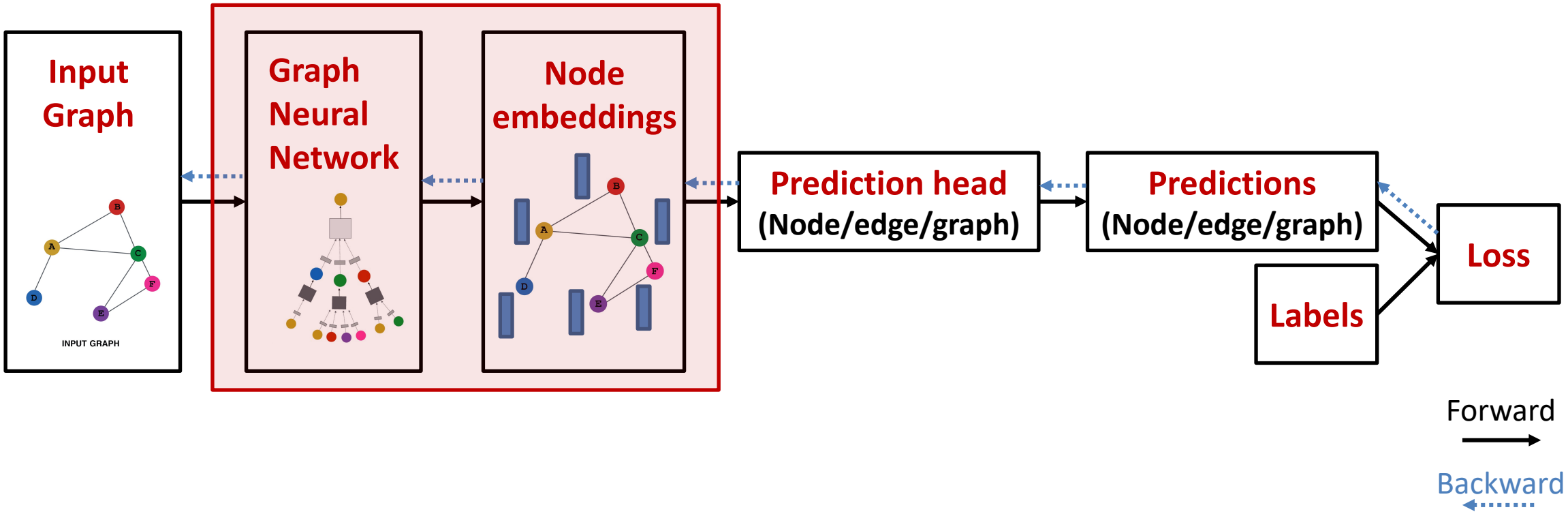
# Deep Learning Pipeline for Graphs




- **Benefits:**
  - Fully data-driven, few hand-engineering
  - Utilize rich feature information in graphs

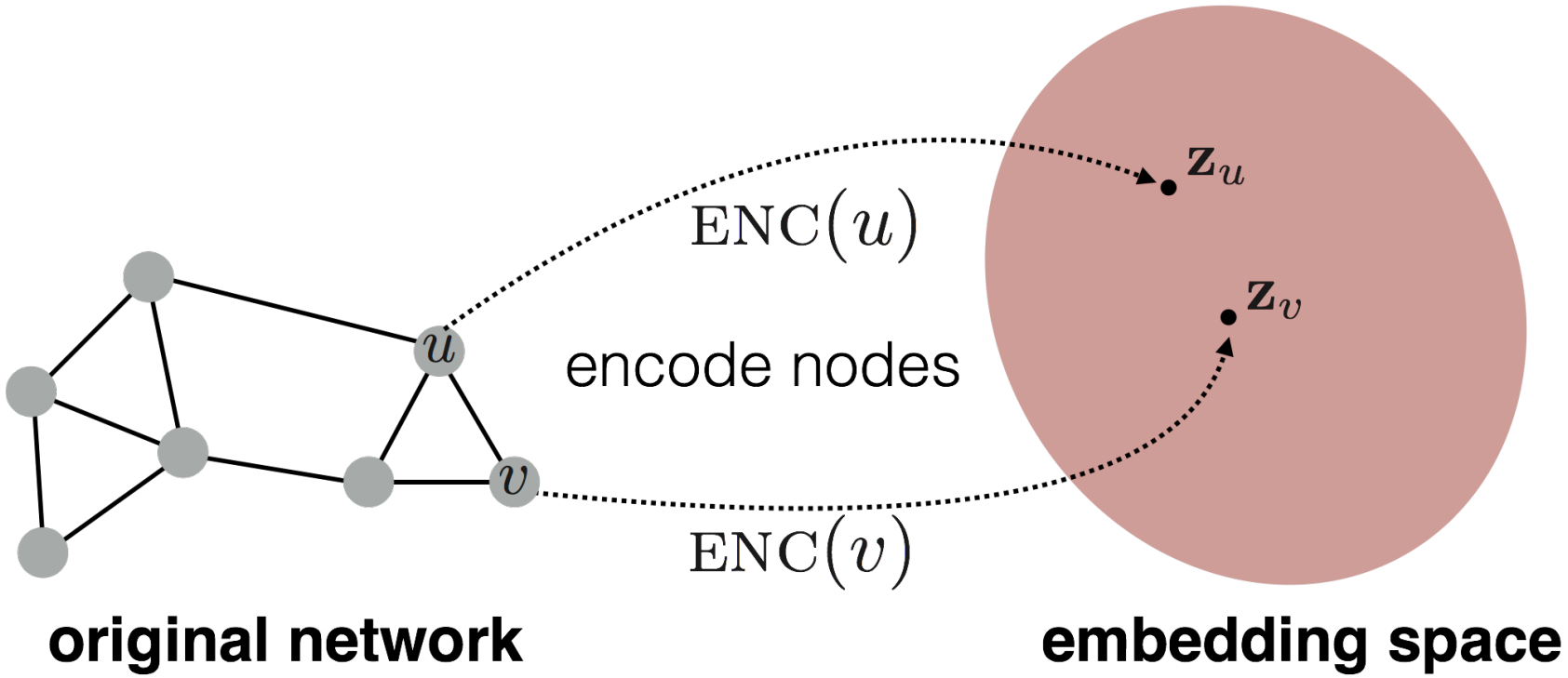
# Deep Learning Pipeline for Graphs

## Key concepts



# Key Concept: Node Embeddings

- **Intuition:** Map nodes to $d$-dimensional embeddings such that similar nodes in the graph are embedded close together

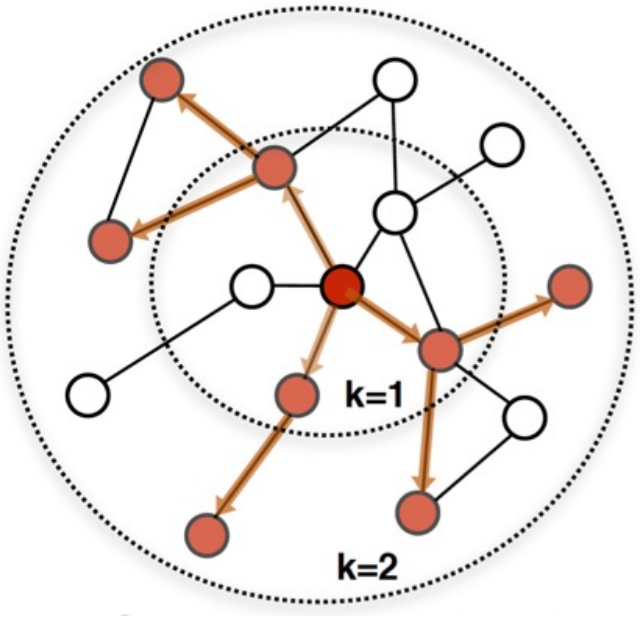



- How to learn the encoder function $ENC(\cdot)$? **GNN!**

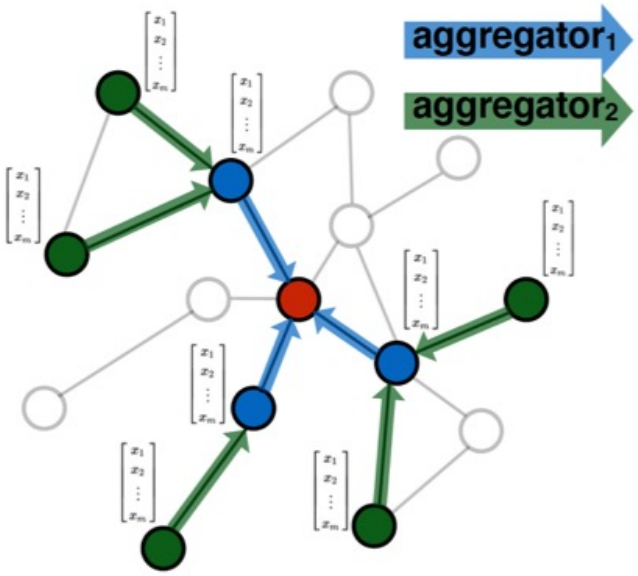
W. Hamilton, R. Ying, J. Leskovec. Inductive Representation Learning on Large Graphs. NeurIPS, 2017

# Key Concept: Graph Neural Network (GNN)

- **Goal:** Learn meaningful node embeddings
- **Idea:** Iteratively aggregate information from a node's neighborhood



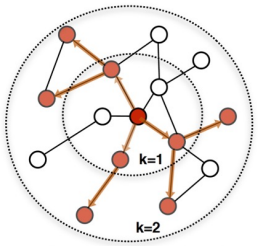

Determine node computation graph

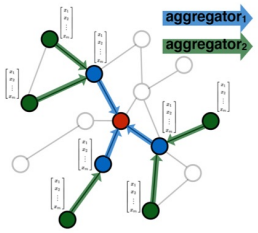



Propagate and transform information

# Overview of This Talk

Graph Learning Basics




Challenges in Applying Graph Learning

- Implementation
- Finding best designs
- Customization

GraphGym

# Challenge 1: Implementing a GNN Pipeline

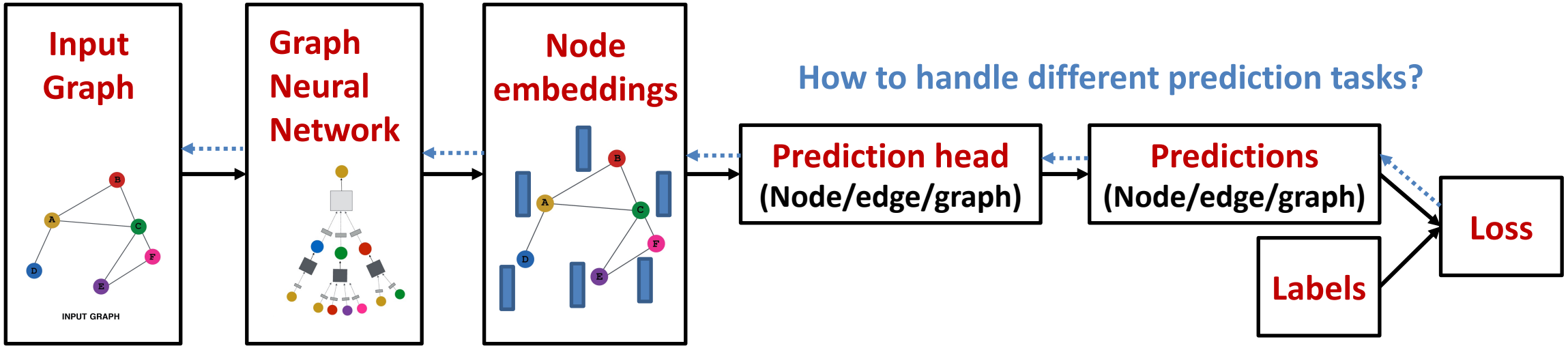
```
from torch_geometric.nn import GCNConv
```

**There is a Gap!**

How to manipulate data?

How to design GNN?

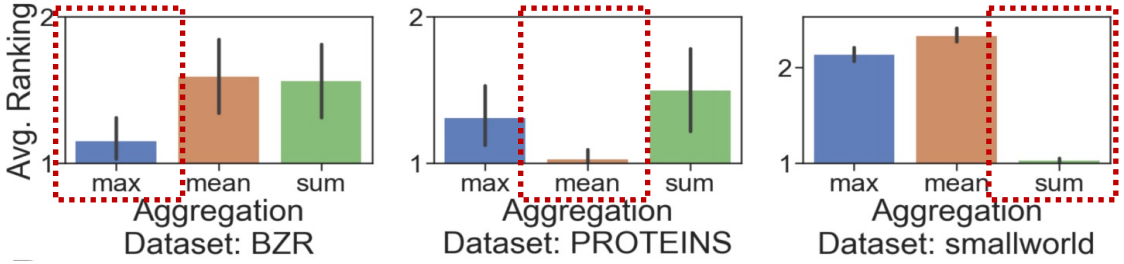
How to handle different prediction tasks?




How to evaluate performance?

# Challenge 2: Finding Best GNN Designs

- **Observation:** The design space for GNN is huge
  - Easily define millions of GNN models
    - *GNN layer:* GCN, GraphSAGE, GAT, …
    - *Aggregation:* Mean, Sum, Max,…
    - *Operators:* BatchNorm? Dropout? L2norm?, …
    - **Exponential** to the design dimensions (e.g., $3^{10} = 59K$ designs)
- **Challenge:** Best GNN designs for different tasks are very different
  - A SOTA GNN in one task may perform badly in another task




| | SOTA on Task A | SOTA on Task $B$: |
|---|---|---|
| GNN design | (1, 8, 3, skipcat, sum) | (1, 4, 2, skipcat, max) |
| **Performance on ogbg-molhiv** | ***0.785*** | ***0.736*** |

# Challenge 3: Customizing a GNN Pipeline

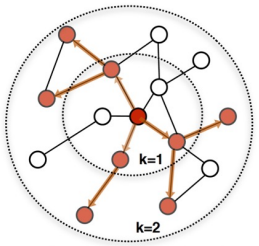
- Suppose you want to apply a new GNN layer: **ABConv**
- **Common workflows:**
  - Fork an open-source pipeline that uses **ABConv**: Repurpose the pipeline to the dataset/task that you need
    - **Issue:** A new pipeline is needed, whenever you want to try something new
  - Adapt your existing GNN pipeline: Replace **GCNConv** with **ABConv**
    - **Issue:** Your core pipeline code needs to be altered, not scalable
- **Common issues:** laborious, inefficient, not scalable, …

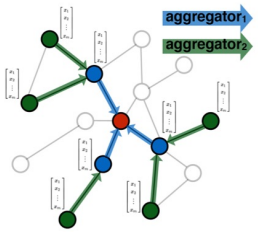
# GraphGym Address The Challenges

- Challenge 1: Implementing a GNN Pipeline
  - **GraphGym** provides a modularized GNN Pipeline
  - Help users explore a giant GNN design space
- Challenge 2: Finding Best GNN Designs
  - **GraphGym** features a simple API for managing GNN experiments
  - Easily launch and analyze thousands of experiments
- Challenge 3: Customizing a GNN Pipeline
  - **GraphGym** allows users to easily register customized modules
  - Increase visibility and impact of SOTA research

# Overview of This Talk

Graph Learning Basics

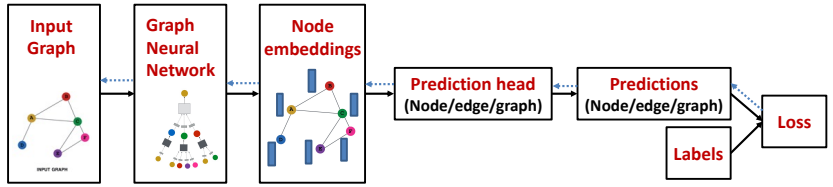



Challenges in Applying Graph Learning

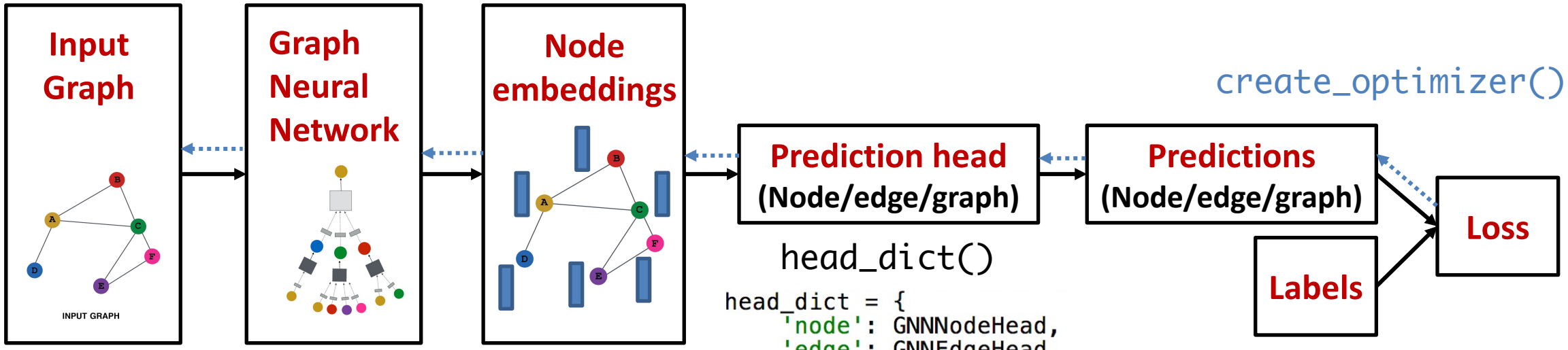
- Implementation
- Finding best designs
- Customization

**GraphGym**




PyG

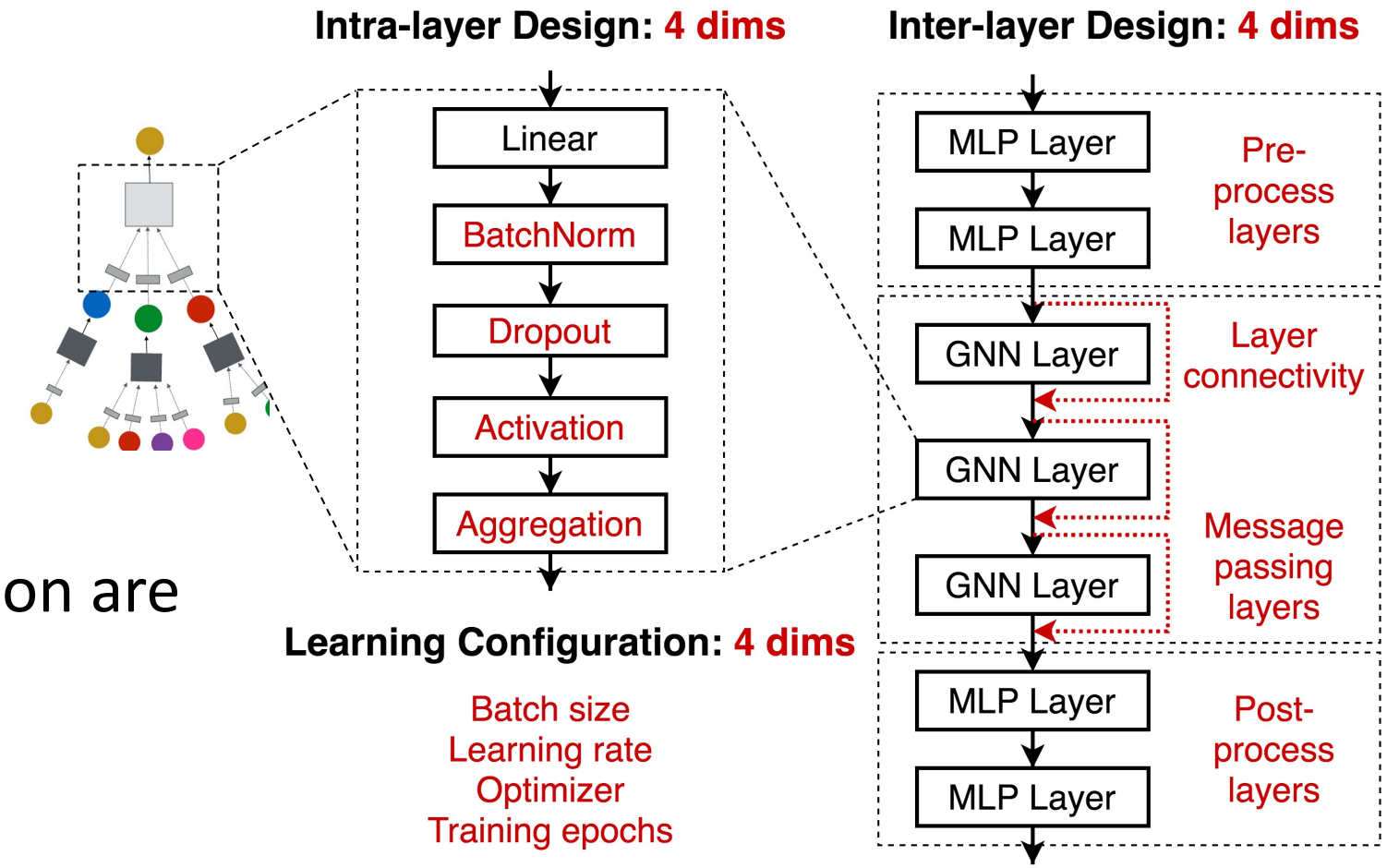
# GraphGym: Modularized GNN Pipeline




```
head_dict = {
    'node': GNNNodeHead,
    'edge': GNNEdgeHead,
    'link_pred': GNNEdgeHead,
    'graph': GNNGraphHead
}
```

# GraphGym: GNN Design Space

- **Core:** A GNN design space
- Main design dimensions
  - Intra-layer design
  - Inter-layer design
  - Learning configuration
  - 315K possible designs
- Many other design dimension are available in GraphGym



# **GraphGym**: Experiment Configuration

```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**

**And more!**

- Each experiment is **fully described by a configuration file**
  - You can **always reproduce** the experiment, **by keeping this config file**

- **Running an experiment (in 1 line!)**

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

# GraphGym: Launching a Batch of Experiments

- **Grid search** over experimental settings

```
# (1) dataset configurations
dataset.format format ['PyG']
dataset.name dataset ['TU_ENZYMES','TU_PROTEINS']
dataset.task task ['graph']
# (2) model configurations
gnn.layers_pre_mp l_pre [1,2]
gnn.layers_mp l_mp [2,4,6,8]
gnn.layers_post_mp l_post [2,3]
gnn.stage_type stage ['skipsum','skipconcat']
gnn.agg agg ['add','mean','max']
```

**Run different datasets**

**Run different models**

- **Launching** GNN experiments in parallel **(in 2 lines!)**
  - Easily scale to **thousands** of experiments (and fully utilize all the GPUs)

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

**Repeat each experiment for 3 random seeds**

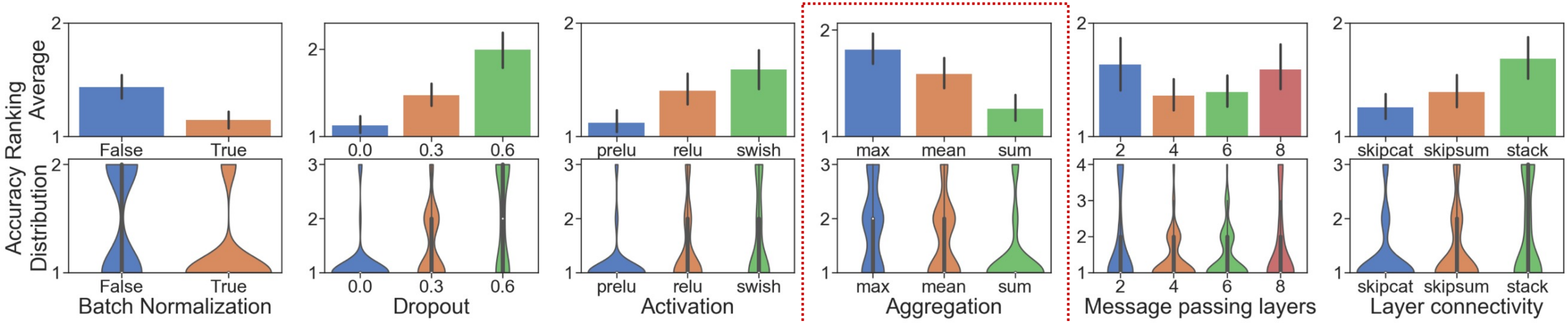
**Run 8 experiments simultaneously**

# GraphGym: Analyze a Batch of Experiments

- **Automatically summarize experiment results and figures**

| l_pre | l_mp | l_post | stage | agg | epoch | loss | loss_std | params | time_iter | time_iter_st | accuracy | accuracy_std |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 2 | skipconcat | add | 399 | 0.5678 | 0.0248 | 217256 | 0.1098 | 0.0075 | 0.886 | 0.0017 |
| 1 | 2 | 2 | skipconcat | max | 399 | 0.3754 | 0.0236 | 217256 | 0.0896 | 0.0026 | 0.9164 | 0.0017 |
| 1 | 2 | 2 | skipconcat | mean | 399 | 0.4885 | 0.0122 | 217256 | 0.0859 | 0.0046 | 0.9083 | 0.0011 |
| 1 | 2 | 2 | skipsum | add | 399 | 0.5624 | 0.022 | 295119 | 0.1121 | 0.0155 | 0.8853 | 0.0039 |
| 1 | 2 | 2 | skipsum | max | 399 | 0.3966 | 0.0054 | 295119 | 0.1049 | 0.003 | 0.9151 | 0.0025 |
| 1 | 2 | 2 | skipsum | mean | 399 | 0.4701 | 0.0118 | 295119 | 0.1027 | 0.0038 | 0.909 | 0.0028 |
| 1 | 2 | 3 | skipconcat | add | 399 | 0.5944 | 0.0231 | 199611 | 0.1138 | 0.0376 | 0.8844 | 0.0082 |

**Aggregated performance**




**Understand the best design: Rank for each design choice**

# GraphGym: Register Customized Modules

```
class ExampleConv(nn.Module):
    def __init__(self, dim_in, dim_out, bias=False, **kwargs):
        super(ExampleConv, self).__init__()
        self.model = ExampleConvLayer(dim_in, dim_out, bias=bias)

    def forward(self, batch):
        batch.x = self.model(batch.x, batch.edge_index)
        return batch

register_layer('exampleconv', ExampleConv)
```

**Register a new GNN layer to GraphGym**

```
gnn.layers_mp l_mp [2,4,6]
gnn.layer_type layer ['gcnconv','exampleconv']
```

**The new ExampleConv layer is ready to be used**

**Supported customized modules:**

- Activations
- Customized configurations
- Feature augmentations
- Feature encoders
- GNN heads
- GNN layers
- Data loaders
- Loss functions
- GNN network architectures
- Optimizers
- GNN global pooling layers
- GNN stages
- GNN training pipelines
- Data transformations

# Application: GraphGym in Research

- **Goal:** Use GraphGym to validate a newly proposed GNN layer (ID-GNN)
- GraphGym can ensure a **fair comparison**:
  - We **fix all the other the hyperparameters** except switching the GNN Layer
  - We **control the computational budget for all the models** to be the same
- We want to show ID-GNN can consistently outperform GCN
  - **4 GNN backbones * 20 node/edge/graph level tasks * 3 random seeds * 2 GNN layers (ID-GNN vs. GCN) = 480 experiments**
- With GraphGym, launching 480 experiments takes 3 lines of code!

```
# 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
#aggregate results for the batch
python agg_batch.py --dir results/baseline_grid_idgnn
```

Run in 10 parallel processes, only taking a few hours to finish

# GraphGym in Domain Applications

- **GraphGym in domain applications:** financial predictions
  - Apply GNN to transaction networks with **millions of nodes and edges**
  - **Knowledge of GraphGym design space + implementation**
- Task 1: Fraud detection
  - **Naïve GNN implementation:** AUC 0.64
  - **GraphGym implementation: AUC 0.94+**
- Task 2: Future transaction prediction
  - **Naïve GNN implementation:** AUC 0.68
  - **GraphGym implementation: AUC 0.88+**

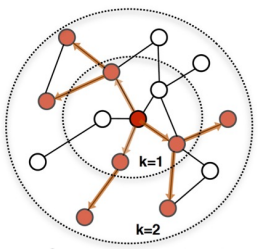
# GraphGym: History and Future

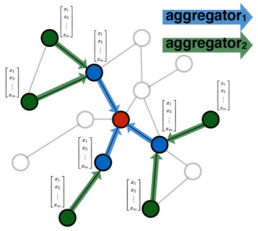
- **GraphGym was originally proposed in 2020**
  - **Paper:** Design Space for Graph Neural Networks, NeurIPS 2020 spotlight
  - **Code:** Originally released at https://github.com/snap-stanford/GraphGym
- **GraphGym has become a core component of PyG 2.0**
  - ```
    pip install torch_geometric
    git clone https://github.com/pyg-team/pytorch_geometric.git
    bash graphgym/run_single.sh  # run a single GNN experiment
    bash graphgym/run_batch.sh  # run a batch of GNN experiments
    ```
  - Excited to be a core PyG team member
  - We are continuously working on better and deeper integration with PyG

PyG
    ```

# Summary of This Talk

Graph Learning Basics

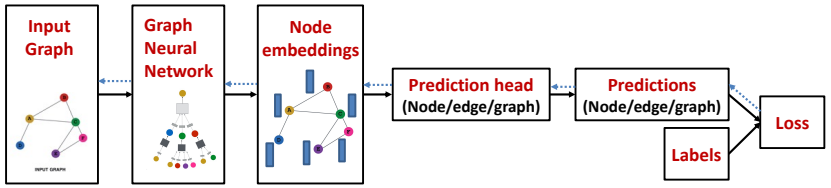



Challenges in Applying Graph Learning

- Implementation
- Finding best designs
- Customization

GraphGym




PyG PyG.org

**Paper:** Design Space for Graph Neural Networks (NeurIPS 2020 spotlight)
**Try it out:** git clone https://github.com/pyg-team/pytorch_geometric.git