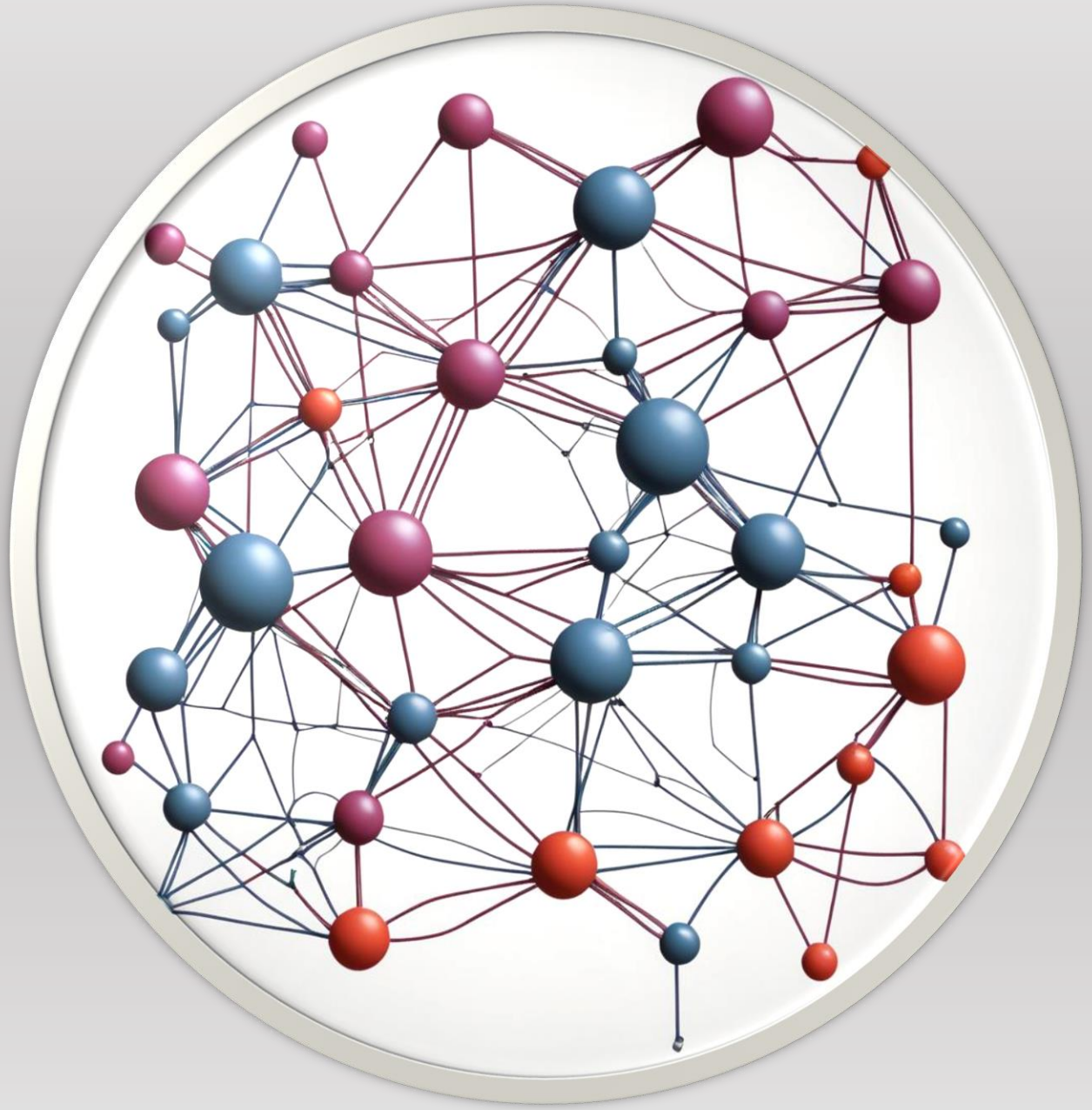


# GraphAny

## A Foundation Model for Node Classification on Any Graph


Ali Vefghi

Supervisor: Z. Rahmati

Author(s): Jianan Zhao et al.

Mila - Québec AI Institute, University of Montréal, Intel AI Lab

University of Oxford, HEC Montréal, CIFAR AI Chair

# Definitions

- **Foundation models**

trained on broad data such that it can be applied across a wide range of use cases (text or images)

a shared input space across all tasks (e.g. a vocabulary of tokens or a patch of pixels)

- **Inference**

- **Inductive/transductive**

- **Homophilic/ heterophilic graphs**

In homophilic graphs, node label largely depends on the labels of its close neighbors whereas in heterophilic graphs, node label does not depend on the neighboring nodes.

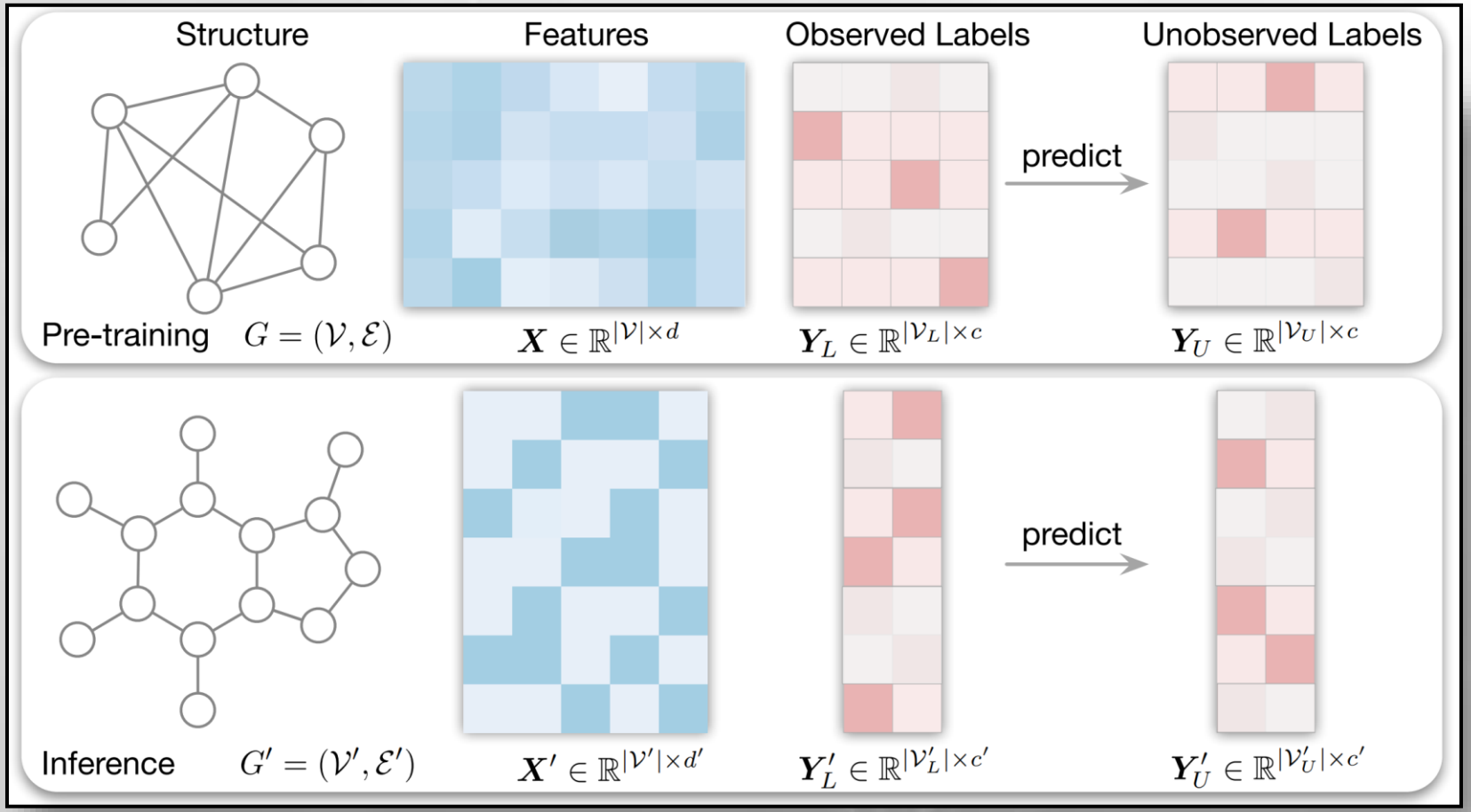
# Problem definition

- a graph G = (V, E)
- an adjacency matrix A
- node features X
- a set of labeled nodes $V_L$
- their labels $Y_L$
- c is the number of unique label classes

The goal is to predict the labels Y' for all the unlabeled nodes

$V_U = V - V_L$

in the graph.



# Problem definition

- **Different feature and label spaces across graphs**

Existing models learn transformations specific to the dimension, type, and structure of the features and labels used in the training, and cannot perform inference on feature and label spaces that are different from the training ones, which can also appear in an arbitrary permuted order.

- **An inductive function**

Existing models learn functions specific to the training graph and cannot generalize to new graphs.

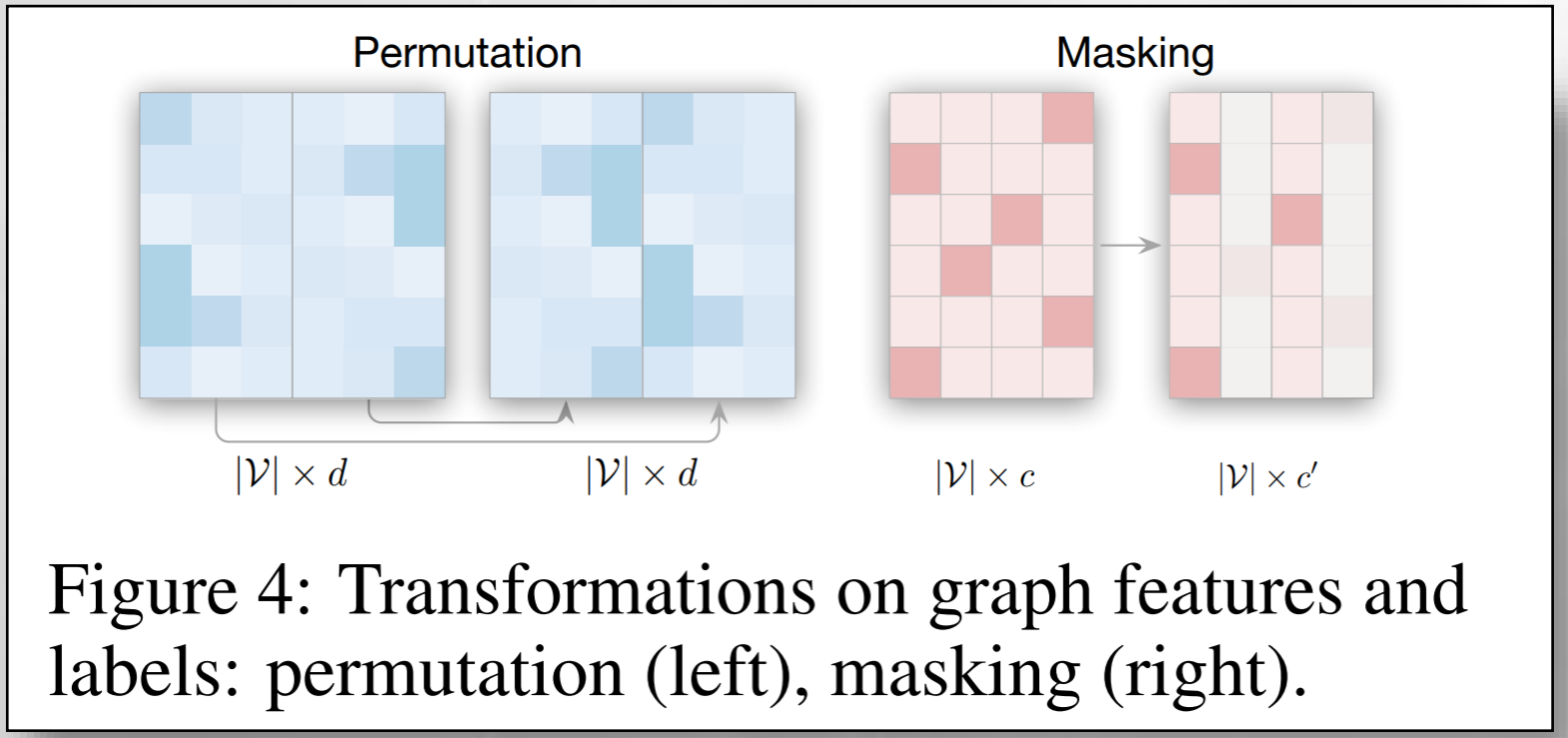



Figure 4: Transformations on graph features and labels: permutation (left), masking (right).

# Related works

- **leverage large language models (LLMs) to solve graph tasks**

- they can only deal with text-attributed graphs (text node features and labels)

- features may even be continuous.

- **Inductive Node Classification**

- use known node labels as features and train-test setup is limited to different partitions of the same bigger graph

- **Labels as Features**

- **Graph Foundation Models**

- the main challenge of graph foundation models (GFM) is finding an invariant graph vocabulary that transfers across different graphs.

- LLMs are used here but the sequential nature of LLMs is suitable for graphs with permutation invariance?

# Method

**GraphAny**

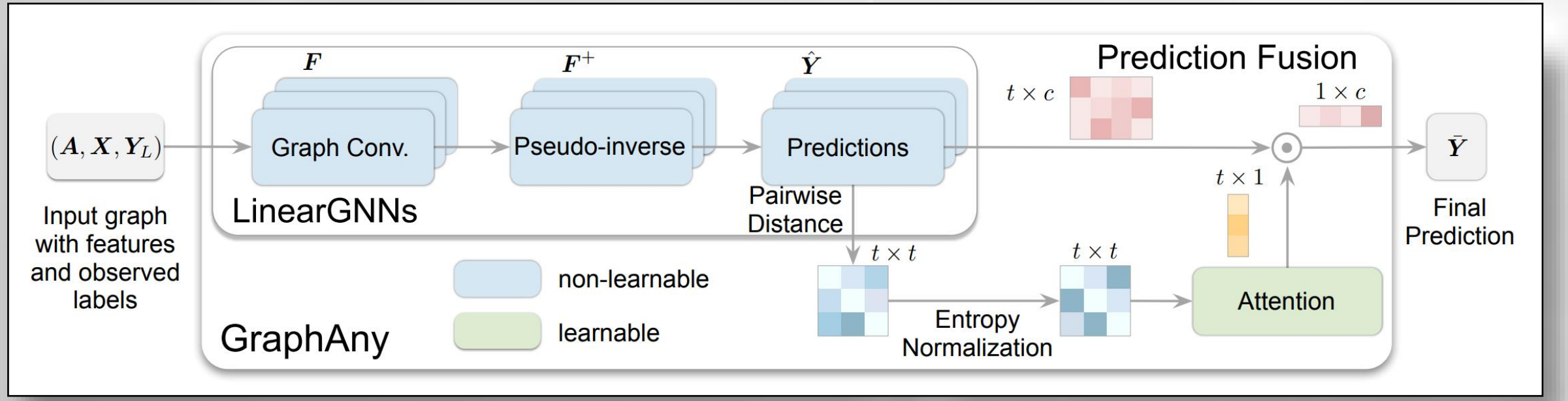
An architecture that can solve node classification on any new graph (homophilic and heterophilic) with any feature and label spaces in both transductive and inductive setups

**Solutions to challenges:**

- models inference on a new graph as an analytical solution to a LinearGNN
- learns attention scores for each node to fuse the predictions of multiple LinearGNNs

- no expensive gradient steps for label prediction

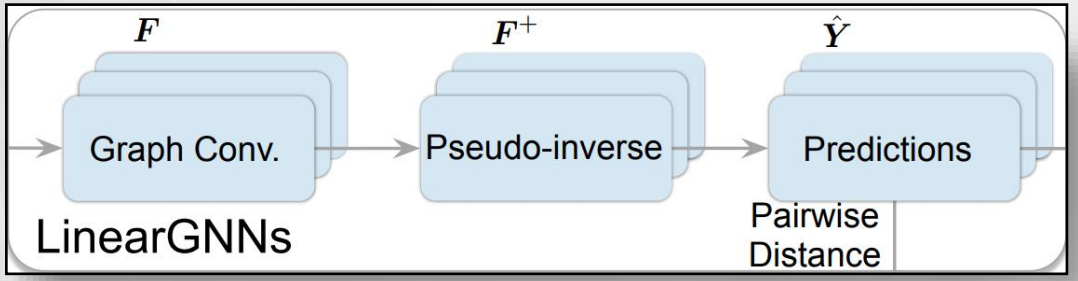
# Method




**Two main components:**

- **LinearGNNs:** An approximation of labels with a transformation in a closed form and non-paramaetric,
- **An attention module:** learns to combine multiple LinearGNNs based on inductive features that generalize to new graphs.

# LinearGNNs




Idea is from simple graph convolutional networks (SGC)

A key idea of this paper is to use simple GNN models whose parameters can be expressed analytically. Among existing models, simple graph convolutional networks (SGC) [44] use a non-parameteric graph convolution operation to process node features, followed by a linear layer to predict the labels,

$$\hat{\boldsymbol{Y}} = \text{softmax}(\boldsymbol{F}\boldsymbol{W}), \tag{1}$$

where $\boldsymbol{F} = \boldsymbol{A}^k\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}|\times d}$ is the processed features and $\boldsymbol{W} \in \mathbb{R}^{d\times c}$ is the weight of the linear layer. Originally, the parameters of SGC were trained to minimize a cross-entropy loss on node classification, which does not have analytical solution and requires gradient steps. However, if we assume the label distribution to be Gaussian, minimizing the cross-entropy loss is equivalent to minimizing the mean-squared error loss [14]:

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \|\hat{\boldsymbol{Y}}_L - \boldsymbol{Y}_L\|^2, \tag{2}$$

where we use $\hat{\boldsymbol{Y}}_L$ to denote model predictions on the set of labeled nodes. The benefit of this approximation is that now we have a closed-form solution for optimal weights $\boldsymbol{W}^*$:

$$\boldsymbol{W}^* = \boldsymbol{F}_L^+\boldsymbol{Y}_L, \tag{3}$$

where $\boldsymbol{F}_L^+$ is the pseudo inverse of $\boldsymbol{F}_L$, and the model prediction is given by:

$$\hat{\boldsymbol{Y}} = \boldsymbol{F}\boldsymbol{F}_L^+\boldsymbol{Y}_L. \tag{4}$$

For full rank $A$ we have the concise formulas

$$A^+ = \begin{cases} (A^*A)^{-1}A^*, & \text{if } \text{rank}(A) = n \le m, \\ A^*(AA^*)^{-1}, & \text{if } \text{rank}(A) = m \le n. \end{cases} \tag{2}$$

Consequently,

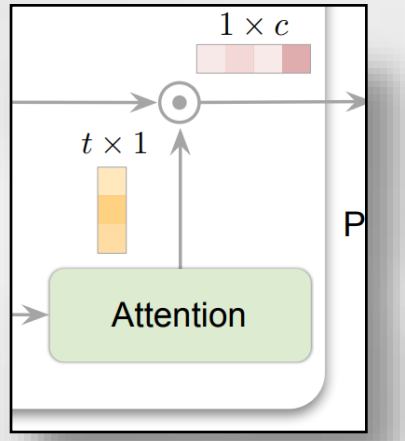
$$A^+A = I_n \text{ if } \text{rank}(A) = n, \qquad AA^+ = I_m \text{ if } \text{rank}(A) = m.$$

$$A^* = \overline{A}^\top = \overline{A^\top}$$

It approximates the prediction of linear GNNs (like SGC) with a single forward pass.

- No training
- Same time complexity as a standard GCN at inference time

# Attention module




An attention module over a set of multiple LinearGNNs

- it should be robust to transformations on features and labels, such as permutation or masking on some dimensions

Let $\alpha_u^{(1)}, \alpha_u^{(2)}, \ldots, \alpha_u^{(t)}$ denote the attention over $t$ LinearGNNs. We generate the final prediction as a combination of all LinearGNN predictions:

$$\bar{\boldsymbol{y}}_u = \sum_{i=1}^{t} \alpha_u^{(i)} \hat{\boldsymbol{y}}_u^{(i)}. \tag{5}$$

# Permutation-Invariant Attention with Distance Features

Our idea is to construct a set of permutation-invariant features, such that any attention module we build on top of these features becomes permutation-invariant. Formally, if the permutation matrices for feature and label dimensions are $\boldsymbol{P} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{Q} \in \mathbb{R}^{c\times c}$ respectively, a function $f$ is *(data) permutation-invariant* if:

$$f(\boldsymbol{X}\boldsymbol{P}, \boldsymbol{Y}_L\boldsymbol{Q}) = f(\boldsymbol{X}, \boldsymbol{Y}_L). \tag{6}$$

In our LinearGNN, the prediction $\hat{\boldsymbol{Y}}$, as a function of the new graph, is invariant to the feature permutation and equivariant to the label permutation since it has the following analytical form:

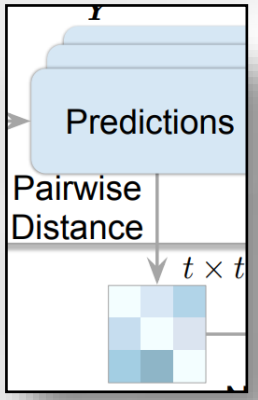
$$\hat{\boldsymbol{Y}}(\boldsymbol{X}\boldsymbol{P}, \boldsymbol{Y}_L\boldsymbol{Q}) = \boldsymbol{F}\boldsymbol{F}_L^{+}\boldsymbol{Y}_L\boldsymbol{Q}. \tag{7}$$

Deriving a feature that is invariant to the label permutation requires us to cancel $\boldsymbol{Q}$ with its inverse $\boldsymbol{Q}^\top$. A straightforward solution is to use a dot product feature for predictions on each node:

$$\hat{\boldsymbol{Y}}\hat{\boldsymbol{Y}}^\top = \boldsymbol{F}\boldsymbol{F}_L^{+}\boldsymbol{Y}_L\boldsymbol{Y}_L^\top(\boldsymbol{F}_L^{+})^\top\boldsymbol{F}^\top, \tag{8}$$

which is invariant to both feature- and label-permutation matrices $\boldsymbol{P}$ and $\boldsymbol{Q}$. Generally, any feature that is a linear combination of dot products between LinearGNN predictions is also permutation invariant, e.g. Euclidean distance or Jensen-Shannon divergence. For a set of LinearGNN predictions $\hat{\boldsymbol{y}}_u^{(1)}, \hat{\boldsymbol{y}}_u^{(2)}, \cdots, \hat{\boldsymbol{y}}_u^{(t)}$ on a single node $u$, we construct the following $t(t-1)$ permutation-invariant features to capture their squared difference:

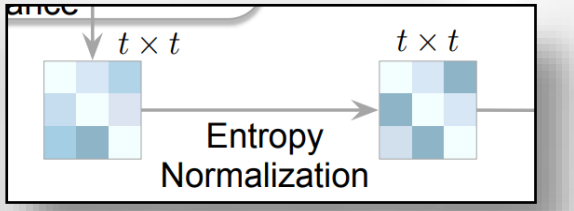
$$\|\hat{\boldsymbol{y}}_u^{(1)} - \hat{\boldsymbol{y}}_u^{(2)}\|^2, \|\hat{\boldsymbol{y}}_u^{(1)} - \hat{\boldsymbol{y}}_u^{(3)}\|^2, ..., \|\hat{\boldsymbol{y}}_u^{(t)} - \hat{\boldsymbol{y}}_u^{(t-1)}\|^2. \tag{9}$$



# Robust Dimension Generalization with Entropy Normalization

To this end, we employ *entropy normalization*, a technique widely used by manifold learning methods [17, 41] to adaptively determine the similarity between vectors. For node $u$, the asymmetric similarity feature between LinearGNN prediction $i$ and $j$ is given by:

$$p_u(j|i) = \frac{\exp(-\|\hat{\boldsymbol{y}}_u^{(i)} - \hat{\boldsymbol{y}}_u^{(j)}\|^2 / 2(\sigma_u^{(i)})^2)}{\sum_{k \neq i} \exp(-\|\hat{\boldsymbol{y}}_u^{(i)} - \hat{\boldsymbol{y}}_u^{(k)}\|^2 / (\sigma_u^{(i)})^2)}, \tag{10}$$




where $\sigma_u^{(i)}$ is the standard deviation of an isotropic multivariate Gaussian distribution, determined by matching the entropy of $p_u(j|i)$ with a hyperparameter $H$. Since the similarity features are functions of the distance features, they are also permutation invariant to the feature and label dimensions of the graph. Intuitively, this imposes a soft constraint on the number of LinearGNN predictions that are regarded as similar to the current prediction $\boldsymbol{y}_u^{(i)}$, which significantly reduces the gap between training and test features. We will verify the effectiveness of entropy normalization in Section 4.4.

# Analysis of Time Complexity

One advantage of GraphAny is that it is more efficient than conventional graph neural networks (e.g. GCN [21], which is due to two reasons. First, LinearGNN leverages non-parameteric graph convolution operations, which can be preprocessed and cached for all nodes on a graph, reducing its inference time complexity to $O(|\mathcal{V}|)$. Second, the analytical solution used by LinearGNN eliminates all gradient steps, leading to a very low optimization cost for any graph $O(|\mathcal{V}_L|)$.

Table 1 shows the time complexity and total wall time of GCN, LinearGNN and GraphAny. The total wall time considers all training and inference time on 31 graphs. Even without any speed optimization in our implementation, GraphAny is 2.95× faster than GCN in total time. We believe the speedup can be even larger with dedicate implementations of GraphAny.

Table 1: Comparison of time complexity and wall time. $P$ is the number of epochs. Note that GCN has to be trained individually on each of the 31 graphs while GraphAny only needs 1 training graph.

| **Model** | **Pre-processing** | **Optimization** | **Inference** | **Total Wall Time** (31 graphs) |
|---|---|---|---|---|
| GCN | 0 | $O(\|\mathcal{E}\|P)$ | $O(\|\mathcal{E}\|)$ | 18.80 min |
| LinearGNN | $O(\|\mathcal{E}\|)$ | $O(\|\mathcal{V}_L\|)$ | $O(\|\mathcal{V}\|)$ | 1.25 min (15.04×) |
| GraphAny | $O(\|\mathcal{E}\|)$ | $O(\|\mathcal{V}_L\|)$ | $O(\|\mathcal{V}\|)$ | 6.37 min (2.95×) |

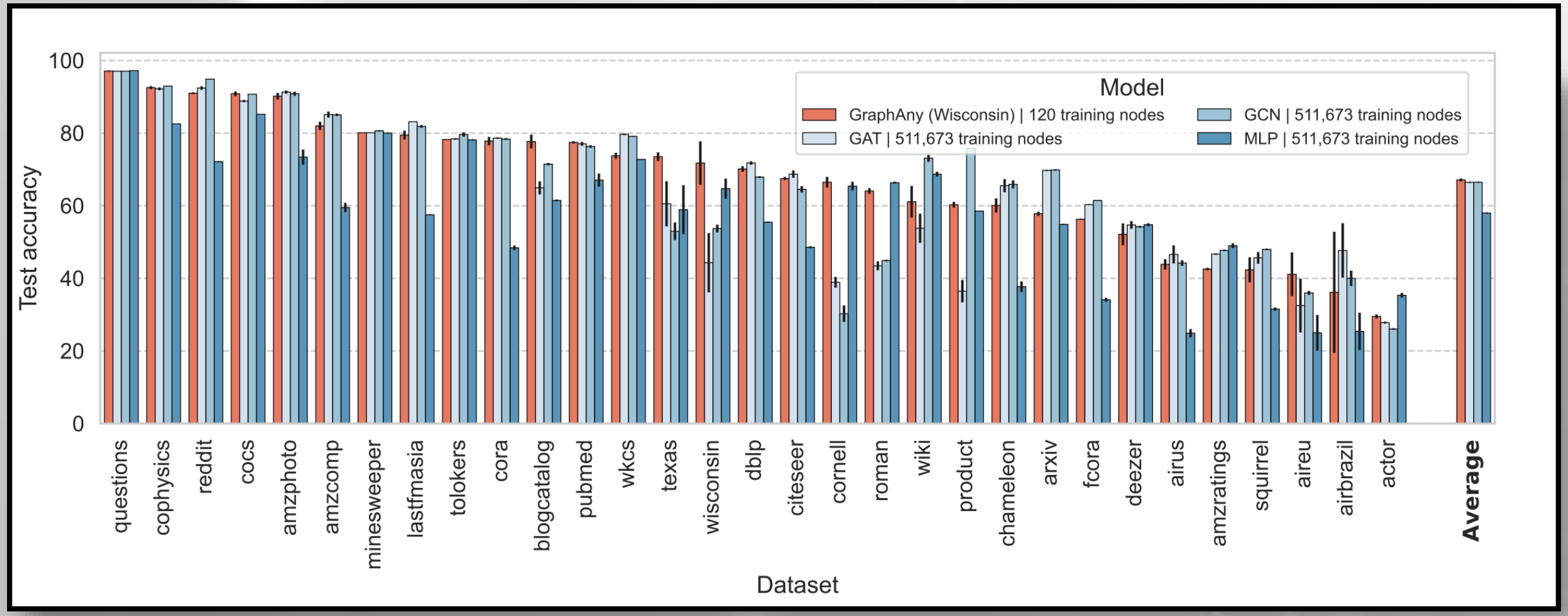
# Datasets

- Compiled a diverse collection of 31 node classification datasets from three sources: PyG , DGL , and OGB .
- These datasets encompass a wide range of graph types including academic collaboration networks, social networks, e-commerce networks and knowledge graphs, with sizes varying from a few hundreds to a few millions of nodes.
- The number of classes across these datasets ranges from 2 to 70.
- default split if it exists, otherwise, standard semi-supervised setting, where 20 nodes are randomly selected as training nodes for each label.

# Implementation details

**Implementation Details.** For GraphAny, we employ 5 LinearGNNs with different graph convolution operations: $\boldsymbol{F} = \boldsymbol{X}$ (Linear), $\boldsymbol{F} = \boldsymbol{A}\boldsymbol{X}$ (LinearSGC1), $\boldsymbol{F} = \boldsymbol{A}^2\boldsymbol{X}$ (LinearSGC2), $\boldsymbol{F} = (\boldsymbol{I} - \boldsymbol{A})\boldsymbol{X}$ (LinearHGC1) and $\boldsymbol{F} = (\boldsymbol{I} - \boldsymbol{A})^2\boldsymbol{X}$ (LinearHGC2), which cover identical, low-pass and high-pass spectral filters. In our experiments, we consider 4 GraphAny models trained separately on 4 datasets respectively: Cora (homophilic, small), Wisconsin (heterophilic, small), Arxiv (homophilic, medium), and Products (homophilic, large). The remaining 27 datasets are held out from these training sets,

# Results


Model
GraphAny (Wisconsin) | 120 training nodes
GAT | 511,673 training nodes
GCN | 511,673 training nodes
MLP | 511,673 training nodes
Test accuracy
100
80
60
40
20
0
Dataset
questions
cophysics
reddit
cocs
amzphoto
amzcomp
minesweeper
lastfmasia
tolokers
cora
blogcatalog
pubmed
wkcs
texas
wisconsin
dblp
citeseer
cornell
roman
wiki
product
chameleon
arxiv
fcora
deezer
airus
amzratings
squirrel
aireu
airbrazil
actor
Average

# Results

Table 2: Main experiment results (test accuracy %).

| Category | Method | Cora | Wisconsin | Arxiv | Products | Held Out Avg. (27 graphs) | Total Avg. (31 graphs) |
|---|---|---|---|---|---|---|---|
| **Transductive** | MLP | 48.42±0.63 | 79.21±1.07 | 55.50±0.23 | 61.06±0.08 | 57.09 | 57.20 |
| | GCN | 81.40±0.70 | 54.51±0.88 | 71.74±0.29 | 75.79±0.12 | 65.55 | 66.55 |
| | GAT | **81.70±1.43** | 52.94±3.10 | **73.65±0.11** | **79.45±0.59** | 65.31 | 67.03 |
| **Non-parameteric** | LabelProp | 60.30±0.00 | 16.08±2.15 | 0.98±0.00 | 74.50±0.00 | 50.73±0.31 | 49.01±0.27 |
| | Linear | 52.80±0.00 | **80.00±2.15** | 46.79±0.00 | 42.10±0.00 | 57.91±0.43 | 57.59±0.42 |
| | LinearSGC1 | 74.30±0.00 | 45.49±13.96 | 55.33±0.00 | 56.58±0.00 | 62.69±0.24 | 62.08±0.48 |
| | LinearSGC2 | 78.20±0.00 | 57.64±1.07 | 59.58±0.00 | 62.92±0.00 | 64.38±0.48 | 64.41±0.39 |
| | LinearHGC1 | 22.50±0.00 | 64.32±2.15 | 22.92±0.00 | 15.00±0.00 | 37.01±0.20 | 36.26±0.23 |
| | LinearHGC2 | 23.80±0.00 | 56.08±4.29 | 20.65±0.00 | 13.39±0.00 | 35.62±0.68 | 34.70±0.55 |
| **Inductive (training set)** | GraphAny (Cora) | 80.18 ±0.13 | 61.18±5.08 | 58.62±0.05 | 61.60±0.10 | 67.24±0.23 | 67.00±0.14 |
| | GraphAny (Wisconsin) | 77.82±1.15 | 71.77±5.98 | 57.79±0.56 | 60.28±0.80 | 67.31±0.38 | 67.26±0.20 |
| | GraphAny (Arxiv) | 79.38±0.16 | 65.10±3.22 | 58.68±0.17 | 61.31±0.20 | 67.65±0.31 | 67.46±0.27 |
| | GraphAny (Products) | 79.36±0.23 | 65.89±2.23 | 58.58±0.11 | 61.19±0.23 | **67.66±0.39** | **67.48±0.33** |

# Results

Table 5: Per-dataset results of all baselines and four GraphAny models

| Dataset | MLP | GCN | GAT | GraphAny (Products) | GraphAny (Arxiv) | GraphAny (Wisconsin) | GraphAny (Cora) |
|---|---|---|---|---|---|---|---|
| Actor | 35.33±0.64 | 26.05±0.23 | 27.82±0.28 | 28.99±0.61 | 28.60±0.21 | 29.51±0.55 | 27.91±0.16 |
| AirBrazil | 25.39±5.16 | 40.00±2.11 | 47.69±7.50 | 34.61±16.54 | 34.61±16.09 | 36.15±16.68 | 33.07±16.68 |
| AirEU | 25.00±4.92 | 36.00±0.56 | 32.50±7.41 | 41.75±6.84 | 41.50±6.50 | 41.13±6.02 | 40.50±7.01 |
| AirUS | 24.90±1.14 | 44.21±0.77 | 46.60±2.49 | 43.57±2.07 | 43.64±1.83 | 43.86±1.44 | 43.46±1.45 |
| AmzComp | 59.53±1.26 | 85.05±0.35 | 85.11±0.84 | 82.90±1.25 | 83.04±1.24 | 82.00±1.14 | 82.99±1.22 |
| AmzPhoto | 73.41±2.08 | 90.86±0.56 | 91.31±0.44 | 90.64±0.82 | 90.60±0.82 | 90.18±0.91 | 90.14±0.93 |
| AmzRatings | 48.99±0.67 | 47.71±0.22 | 46.70±0.21 | 42.70±0.10 | 42.74±0.12 | 42.57±0.34 | 42.84±0.04 |
| ogbn-arxiv | 54.87±0.15 | 69.87±0.18 | 69.75±0.17 | 58.58±0.11 | 58.68±0.17 | 57.79±0.56 | 58.62±0.05 |
| BlogCatalog | 61.47±0.25 | 71.46±0.32 | 64.93±1.81 | 74.73±3.19 | 73.63±2.95 | 77.69±1.90 | 72.52±3.22 |
| Chameleon | 37.72±1.45 | 65.88±1.11 | 65.53±1.81 | 62.59±0.87 | 62.59±0.86 | 60.09±1.93 | 61.49±1.88 |
| Citeseer | 48.56±0.27 | 64.52±0.89 | 68.72±1.01 | 67.94±0.29 | 68.34±0.23 | 67.50±0.44 | 68.90±0.07 |
| CoCS | 85.20±0.10 | 90.72±0.05 | 88.84±0.31 | 90.46±0.54 | 90.45±0.59 | 90.85±0.63 | 90.47±0.63 |
| CoPhysics | 82.57±0.08 | 92.96±0.05 | 92.21±0.39 | 92.66±0.52 | 92.69±0.52 | 92.54±0.43 | 92.70±0.54 |
| Cora | 48.42±0.63 | 78.36±0.28 | 78.64±0.21 | 79.36±0.23 | 79.38±0.16 | 77.82±1.15 | 80.18±0.13 |
| Cornell | 65.40±1.21 | 30.27±2.26 | 38.92±1.48 | 64.86±0.00 | 65.94±1.48 | 66.49±1.48 | 64.86±1.91 |
| DBLP | 55.47±0.18 | 67.90±0.24 | 71.78±0.48 | 70.62±0.97 | 70.90±0.88 | 70.13±0.77 | 71.73±0.94 |
| Deezer | 54.78±0.42 | 54.22±0.27 | 54.71±1.06 | 52.09±2.78 | 52.11±2.79 | 52.13±3.02 | 51.98±2.79 |
| FullCora | 34.12±0.53 | 61.46±0.07 | 60.31±0.14 | 57.13±0.37 | 57.25±0.43 | 56.29±0.17 | 56.73±0.41 |
| LastFMAsia | 57.51±0.16 | 81.85±0.38 | 83.15±0.05 | 80.17±0.44 | 80.60±0.58 | 79.47±1.23 | 80.83±0.41 |
| Minesweeper | 80.00±0.00 | 80.64±0.19 | 80.13±0.05 | 80.27±0.16 | 80.30±0.13 | 80.13±0.09 | 80.46±0.15 |
| ogbn-products | 58.54±0.04 | 75.79±0.12 | 36.47±3.09 | 61.19±0.23 | 61.31±0.20 | 60.28±0.80 | 61.60±0.10 |
| Pubmed | 67.06±1.81 | 76.32±0.37 | 77.06±0.51 | 76.54±0.34 | 76.36±0.17 | 77.46±0.30 | 76.60±0.31 |
| Questions | 97.23±0.02 | 97.07±0.01 | 97.06±0.02 | 97.10±0.01 | 97.09±0.02 | 97.11±0.00 | 97.06±0.03 |
| Reddit | 72.17±0.11 | 94.87±0.02 | 92.43±0.48 | 90.67±0.13 | 90.58±0.12 | 91.00±0.24 | 90.46±0.03 |
| RomanEmpire | 66.35±0.28 | 44.91±0.24 | 43.46±1.23 | 64.66±0.84 | 64.25±1.09 | 64.06±0.78 | 64.25±0.64 |
| Squirrel | 31.55±0.44 | 47.99±0.22 | 45.65±1.61 | 49.45±0.67 | 49.70±0.95 | 42.34±3.46 | 48.49±0.98 |
| Texas | 58.92±6.73 | 52.97±2.42 | 60.54±6.22 | 73.52±2.96 | 72.97±2.71 | 73.51±1.21 | 71.89±1.48 |
| Tolokers | 78.16±0.00 | 79.63±0.58 | 78.41±0.22 | 78.18±0.03 | 78.18±0.04 | 78.24±0.03 | 78.20±0.02 |
| Wiki | 68.71±0.66 | 73.11±0.97 | 53.81±4.02 | 63.08±3.61 | 62.96±3.68 | 61.10±4.36 | 60.56±3.62 |
| Wisconsin | 64.71±2.78 | 53.72±1.07 | 44.31±8.16 | 65.89±2.23 | 65.10±3.22 | 71.77±5.98 | 61.18±5.08 |
| WikiCS | 72.78±0.09 | 79.11±0.14 | 79.68±0.22 | 75.01±0.54 | 74.95±0.61 | 73.77±0.83 | 74.39±0.71 |

# Results

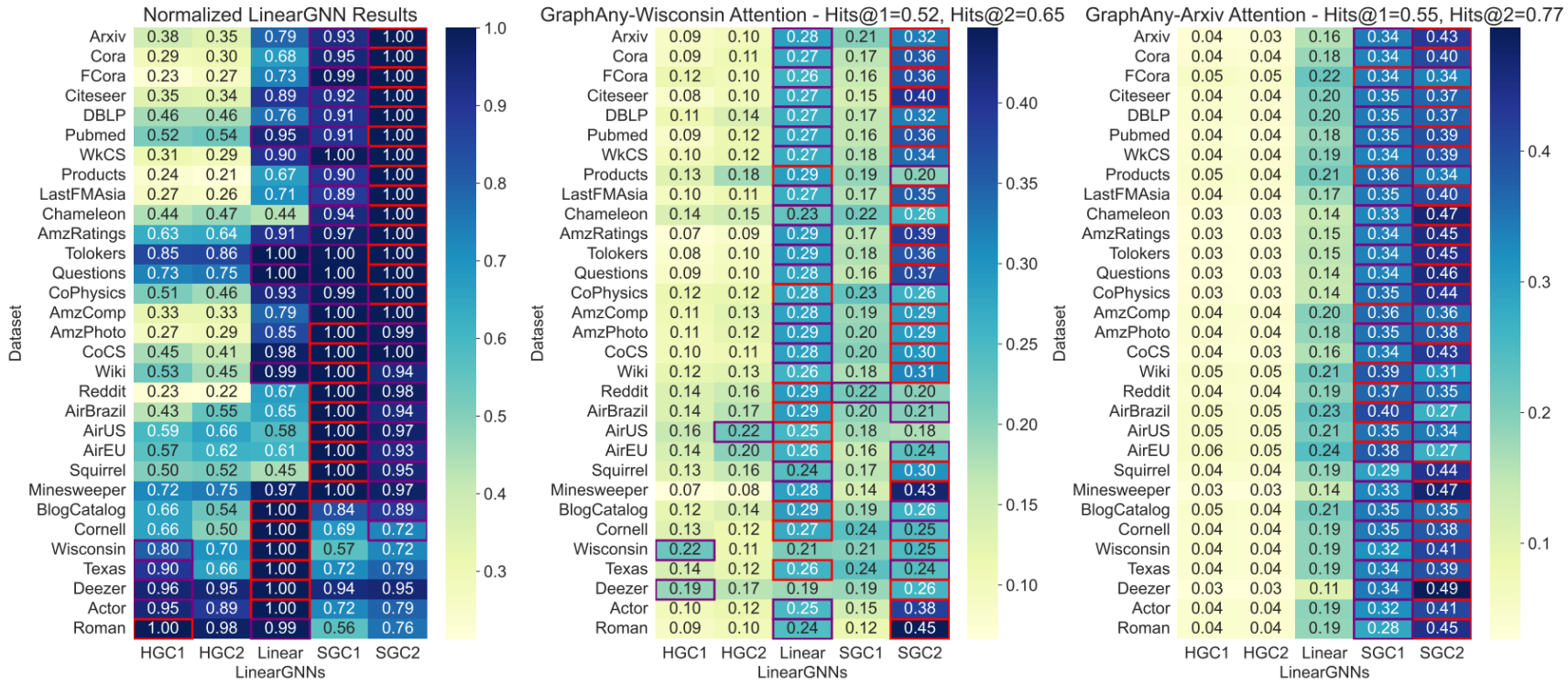



Figure 6: Normalized performance of LinearGNNs (left, best as 1) and attention weights of GraphAny trained on Wisconsin (middle) and Arxiv (right) respectively. The best and second best LinearGNN performance and attention weights for each dataset are highlighted with red and purple rectangles respectively. Our inductive attention can identify the best-performing LinearGNN for most datasets.

# Ablation Study on Entropy Normalization and Attention Parameterization

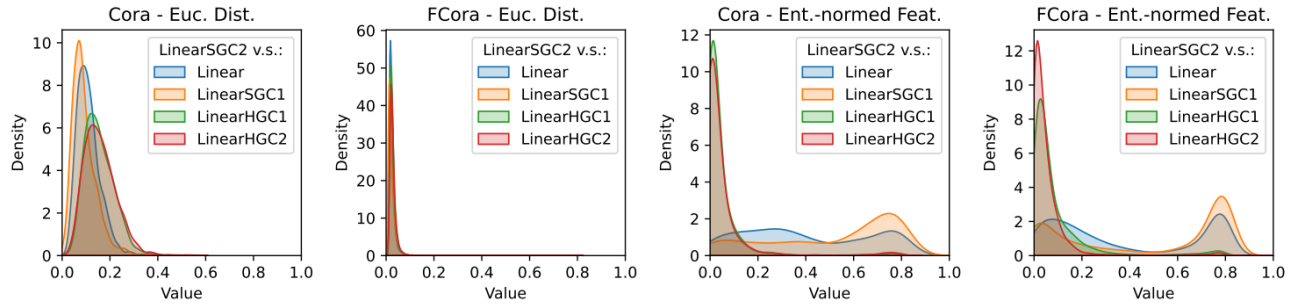



Figure 7: Cora (7 classes) vs Full Cora (70 classes). Left two plots: Without entropy normalization, there is a significant disparity in feature distributions between datasets. Right two plots: With entropy normalization applied, the disparity in feature distributions across datasets is largely reduced.

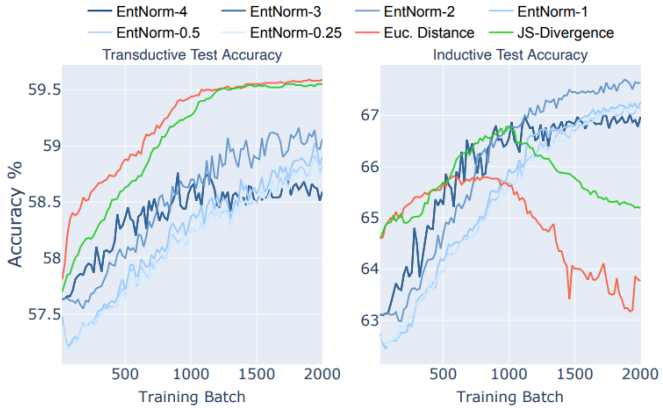



Figure 8: Performance of different distance features with and without entropy normalization.

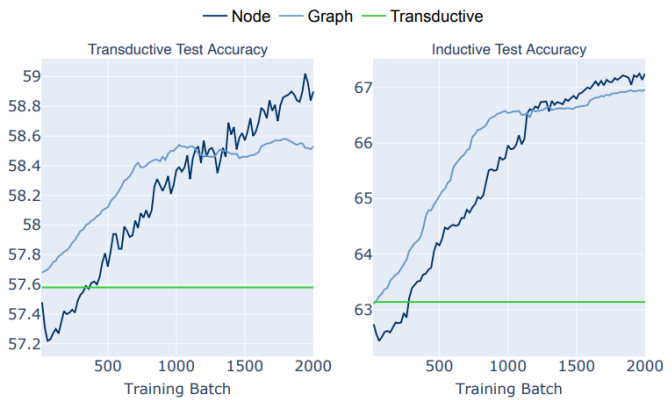



Figure 9: Performance of different attention parameterizations.

# Limitations and Future Work

- First, LinearGNNs have limited expressiveness and cannot fit complex functions on graphs, which may restrict the overall expressiveness of GraphAny even if we combine LinearGNNs with learned attention for each node
- Second, GraphAny can only solve node classification tasks at the moment, excluding regression tasks, edge-level or graph-level tasks
- Besides, it remains an open question how to apply GraphAny to relational graphs, since non-pararmetric models have never been shown effective for relational graphs
- What did I learn from this paper?


Thank You