

GraphAny: A Foundation Model for Node Classification on Any Graph

Jianan Zhao^{1,2}, Hesham Mostafa³, Mikhail Galkin³, Michael Bronstein⁴
Zhaocheng Zhu^{1,2}, Jian Tang^{1,5,6}

¹Mila - Québec AI Institute, ²University of Montréal, ³Intel AI Lab

⁴University of Oxford, ⁵HEC Montréal, ⁶CIFAR AI Chair

Abstract

Foundation models that can perform inference on any new task without requiring specific training have revolutionized machine learning in vision and language applications. However, applications involving graph-structured data remain a tough nut for foundation models, due to challenges in the unique feature- and label spaces associated with each graph. Traditional graph ML models such as graph neural networks (GNNs) trained on graphs cannot perform inference on a new graph with feature and label spaces different from the training ones. Furthermore, existing models learn functions specific to the training graph and cannot generalize to new graphs. In this work, we tackle these two challenges with a new foundational architecture for inductive node classification named GraphAny. GraphAny models inference on a new graph as an analytical solution to a LinearGNN, thereby solving the first challenge. To solve the second challenge, we learn attention scores for each node to fuse the predictions of multiple LinearGNNs. Specifically, the attention module is carefully parameterized as a function of the entropy-normalized distance-features between multiple LinearGNNs predictions to ensure generalization to new graphs. Empirically, GraphAny trained on the Wisconsin dataset with only 120 labeled nodes can effectively generalize to 30 new graphs with an average accuracy of 67.26% in an inductive manner, surpassing GCN and GAT trained in the supervised regime, as well as other inductive baselines.

1 Introduction

Foundation models [1, 39, 40] pre-trained on massive data have drastically changed the landscape of artificial intelligence by their ability to solve any new task conditioned on just a few demonstrations [8, 3]. So far, this ability has mainly been limited to modalities like text or images. An intrinsic property of these modalities is the presence of a *shared input space* across all tasks (e.g. a vocabulary of tokens or a patch of pixels), providing a natural basis for learning a single foundation model over a broad class of different tasks and generalizing to new ones. Graphs, however, do not possess such a property. Taking the node classification task as an example, each graph may have unique dimensions and semantics for its feature and label spaces (e.g. continuous or discrete), which prevents us from developing *graph foundation models* [26] in the same way as for the above modalities.

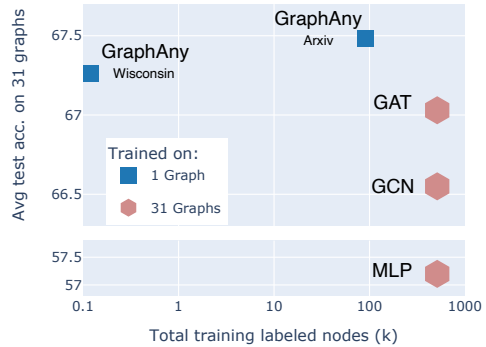


Figure 1: Average performance on 31 datasets. GraphAny is trained on a single dataset (Wisconsin or Arxiv) and performs inductive inference on any graph. The other transductive methods need to be trained on each dataset.

Recently, several attempts have been made to leverage large language models (LLMs) to solve graph tasks [50, 38, 9]. While these methods demonstrate different ways of incorporating graphs into existing LLMs, they can only deal with text-attributed graphs, a subset of graphs whose node features and labels can be easily verbalized as textual sequences and fed into an LLM. For general graphs, node features and labels often do not have textual descriptions, and features may even be continuous, which often renders LLM-based approaches inadequate as graph foundational models.

Therefore, we are interested in an architecture that can *solve node classification on any new graph with any feature and label spaces* in a “native” manner. Such a goal is challenging for existing models on graphs due to two reasons: (1) 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. This requires us to develop a new model architecture for arbitrary feature and label spaces. (2) Existing models learn functions specific to the training graph and cannot generalize to new graphs; this calls for an inductive function that can generalize to any graph once it is trained.

Our Contributions In this work, we present GraphAny, a novel pre-trained foundation model architecture that can solve node classification on any new graph. GraphAny consists of two components: a *LinearGNN* that performs inference on new feature and label spaces without training steps, and an *attention vector* for each node based on entropy-normalized distance features that ensure generalization to new graphs. Specifically, our LinearGNN models the mapping between node features and labels as a non-parameteric graph convolution followed by a linear layer, whose parameters are determined in closed form without requiring explicit training steps. While a single LinearGNN model may be far from optimal for many graphs, we employ multiple LinearGNN models with different graph convolution operators and learn an attention vector to fuse their predictions. The attention vector is carefully parameterized as a function of *distance features* between the predictions of LinearGNNs, which guarantees the model to be invariant to the permutations of feature and label dimensions. To further improve the generalization ability of our model, we propose entropy normalization to rectify the distance feature distribution to a fixed entropy, which reduces the effect of different label dimensions on distance features. Intuitively, the attention vector learns to select the most effective combination of LinearGNNs for each node based on their prediction distributions, reflects statistics of its local structure (e.g., homophily metrics [25]), and generalizes to new graphs.

Empirically, we show that GraphAny trained on any single node classification dataset can generalize to 30 new graphs of different feature and label spaces, even surpassing the average performance of GCN and GAT baselines supervisedly trained on each dataset. We attribute this surprising performance to the attention function GraphAny learns to combine LinearGNNs, which generalizes to new graphs thanks to our careful parameterization. By visualizing the attention in GraphAny, we provide insights into how the attention selects the optimal LinearGNN and how it is affected by the choice of the training graph. We further conduct ablation studies on our entropy normalization and attention parameterization to verify their effectiveness in learning inductive attention. To summarize, the contribution of GraphAny are three folds:

- We propose LinearGNN, an efficient architecture for inductive node classification on any graph with arbitrary feature and label spaces.
- We devise an inductive attention module to fuse LinearGNNs using distance features and entropy normalization, which is invariant to feature and label permutations and robust to dimension changes.
- Combining LinearGNNs and the inductive attention module, GraphAny trained on a single dataset can generalize to any graph, enabling the first graph foundation model for node classification.

2 Related Work

Inductive Node Classification. It is common to categorize prediction tasks into *transductive* and *inductive* setups. In the transductive (semi-supervised) node classification setup, test nodes belong to the same training graph the model was trained on. In the inductive setup, the test graph might be different. The majority of GNNs aiming to work in the inductive setup use known node labels as features. Hang et al. [16] introduced *Collective Learning* GNNs with iterative prediction refinement. Jang et al. [20] applied a diffusion model for such prediction refinement. Structured Proxy Networks [31] combined GNNs and conditional random fields (CRF) in the *structured prediction*

framework. However, the train-test setup in all those works is limited to different partitions of the same bigger graph, that is, they cannot generalize to unseen graphs with different input feature dimensions and number of classes. To the best of our knowledge, GraphAny is the first approach for inductive node classification to do so.

Labels as Features. Addanki et al. [2] demonstrated that node labels used as features yield noticeable improvements in the transductive node classification setup on MAG-240M in OGB-LSC [18]. Sato [34] used labels as features for training-free transductive node classification with a specific GNN weight initialization strategy mimicking label propagation (hence only applicable to **homophilic** 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. GraphAny supports both homophilic and heterophilic graphs in both transductive and inductive setups.

Graph Foundation Models. Inspired by the progress in language and vision foundation models, the idea of having a single model transferable to an arbitrary graph and, possibly, to different graph-level tasks, has recently been gaining increased attention in the graph learning community. Mao et al. [26] identify the main challenge of *graph foundation models* (GFM) as finding an invariant *graph vocabulary* that transfers across different graphs. A variety of methods [50, 38, 9, 11, 29] resort to LLMs as universal featurizers by verbalizing the (sub)graph structure and corresponding task as natural language text. Liu et al. [24] further add a GNN on top of LLM features. However, it is unclear whether the sequential nature of LLMs is suitable for graphs with permutation invariance.

Leveraging GNNs as a backbone of GFMs has been found promising in molecular learning [22, 37], geometric learning for crystal structures [36], and machine learning potentials [4, 23, 48, 45] where a unified graph vocabulary exists, that is, atoms in the periodic table. In non-featurized graphs, GNNs with *labeling tricks* [49] serve as generalizable link predictors in homogeneous graphs [10] and multi-relational knowledge graphs [13]. GFMs for node classification on featurized graphs present a challenge due to heterogeneous feature and label spaces – GraphAny is the first approach to tackle this problem where a single model can perform node classification on unseen homophilic and heterophilic graphs with various feature types and dimensions and number of labels.

3 GraphAny: A Foundation Model for Node Classification

Our goal is to devise a graph foundation model that can perform inductive inference on any new graph with arbitrary feature and label spaces, typically different from the ones associated with the training graph (Figure 2). Here we propose such a solution GraphAny, which consists of two main components (Figure 3): a LinearGNN and an attention module. LinearGNNs provide a basic solution to inductive inference on new graphs with arbitrary feature and label spaces, while the attention module learns to combine multiple LinearGNNs based on inductive features that generalize to new graphs.

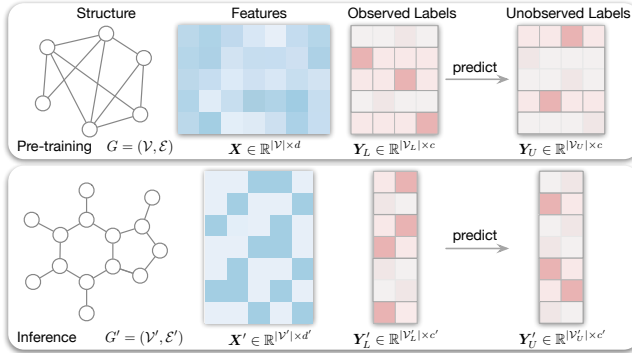


Figure 2: The goal of graph foundation model (GFM) for node classification: A GFM trained on a pre-training graph G should be able to perform inference on any new graph G' with new feature and label spaces.

In a semi-supervised node classification task, we are given a graph $G = (\mathcal{V}, \mathcal{E})$, typically represented as an adjacency matrix A , and node features $X \in \mathbb{R}^{|\mathcal{V}| \times d}$, a set of labeled nodes \mathcal{V}_L and their labels $Y_L \in \mathbb{R}^{|\mathcal{V}_L| \times c}$, where c is the number of unique label classes. The goal of node classification is to predict the labels \hat{Y} for all the unlabeled nodes $\mathcal{V}_U = \mathcal{V} \setminus \mathcal{V}_L$ in the graph. In the conventional transductive learning setup, this is performed by training a GNN (e.g. GCN [21], GAT [42]) on the subset of labeled nodes using standard backpropagation requiring multiple gradient steps. Such a GNN assumes the full graph to be given and typically does not generalize to a new graph “out of the box” without re-training or at least fine-tuning. Conversely, a graph foundation model is expected to predict the labels \hat{Y} for any graph without expensive gradient steps. Furthermore, when a new graph

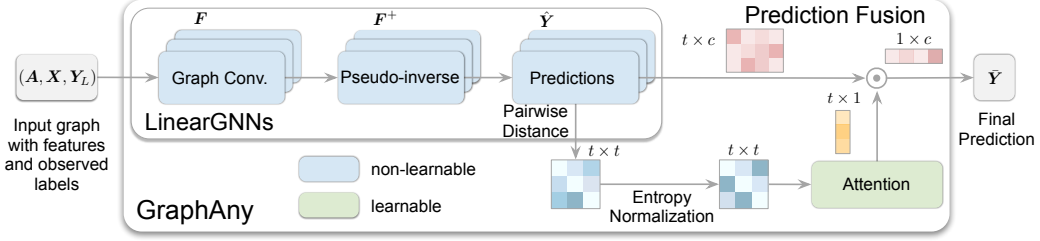


Figure 3: Overview of GraphAny: LinearGNNs are used to perform non-parametric predictions and derives the entropy-normalized distance features. The final prediction is generated by fusing multiple LinearGNN predictions on each node with an attention learned based on the distance features.

is provided, it might have different dimensionality d' of the features and number of class labels, c' , which can also appear in an arbitrary permuted order.

3.1 LinearGNNs for Inductive Inference on Any Graph

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-parametric graph convolution operation to process node features, followed by a linear layer to predict the labels,

$$\hat{\mathbf{Y}} = \text{softmax}(\mathbf{F}\mathbf{W}), \quad (1)$$

where $\mathbf{F} = \mathbf{A}^k \mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the processed features and $\mathbf{W} \in \mathbb{R}^{d \times c}$ 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]:

$$\mathbf{W}^* = \arg \min_{\mathbf{W}} \|\hat{\mathbf{Y}}_L - \mathbf{Y}_L\|^2, \quad (2)$$

where we use $\hat{\mathbf{Y}}_L$ 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 \mathbf{W}^* :

$$\mathbf{W}^* = \mathbf{F}_L^+ \mathbf{Y}_L, \quad (3)$$

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

$$\hat{\mathbf{Y}} = \mathbf{F} \mathbf{F}_L^+ \mathbf{Y}_L. \quad (4)$$

We term this architecture *LinearGNN*, as it approximates the prediction of linear GNNs (like SGC) with a single forward pass. Its advantage is that it does not require training and have the same time complexity as a standard GCN at inference time (see Section 3.3). Note that the derivation of LinearGNNs is independent of the graph convolution operation, and can be applied to other linear convolution operations as well. We stress that while we don't expect LinearGNNs to outperform existing transductive models on node classification, they provide a simple basic module for inductive inference. In Section 3.2, we will see how to combine multiple LinearGNNs to create a stronger model with inductive attention that generalizes to new graphs.

3.2 Learning Inductive Attention over LinearGNNs

LinearGNNs provide a basic solution to inductive inference on new graphs, but they do not learn functions from their training graphs. Besides, our experiments (Figure 6) suggest that different graphs may require LinearGNNs with different convolution operations. Hence, a natural way to incorporate learning in GraphAny is to add an *attention module* over a set of multiple LinearGNNs. Let $\alpha_u^{(1)}, \alpha_u^{(2)}, \dots, \alpha_u^{(t)}$ denote the attention over t LinearGNNs. We generate the final prediction as a combination of all LinearGNN predictions:

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

While there are various ways to parameterize this attention module, finding an inductive solution is non-trivial since neural networks can easily fit some information specific to the training graph. We notice a necessary condition for an inductive attention is that it should be robust to transformations on features and labels, such as permutation or masking on some dimensions (Figure 4). This requires our attention module to be permutation invariant and robust to dimension changes, which motivates our design of distance features and entropy normalization respectively.

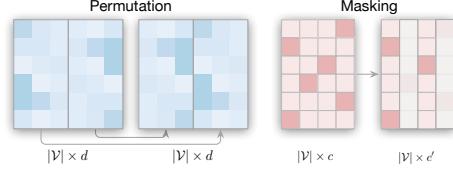


Figure 4: Transformations on graph features and labels: permutation (left), masking (right).

Permutation-Invariant Attention with Distance Features. We would like to design an attention module that is permutation invariant [7] along the data dimension.¹ Consider a new graph generated by permuting feature and label dimensions of the training graph. We expect our attention output to be invariant to these permutations in order to generate the same prediction as for the training set.

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 $P \in \mathbb{R}^{d \times d}$ and $Q \in \mathbb{R}^{c \times c}$ respectively, a function f is (*data*) *permutation-invariant* if:

$$f(XP, Y_LQ) = f(X, Y_L). \quad (6)$$

In our LinearGNN, the prediction \hat{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:

$$\hat{Y}(XP, Y_LQ) = FF_L^+ Y_LQ. \quad (7)$$

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

$$\hat{Y}\hat{Y}^\top = FF_L^+ Y_L Y_L^\top (F_L^+)^T F^\top, \quad (8)$$

which is invariant to both feature- and label-permutation matrices P and 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 $\hat{y}_u^{(1)}, \hat{y}_u^{(2)}, \dots, \hat{y}_u^{(t)}$ on a single node u , we construct the following $t(t-1)$ permutation-invariant features to capture their squared difference:

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

We include detailed proofs in Appendix A. An advantage of such permutation-invariant features is that any model taking them as input is also permutation invariant, allowing us to use a simple model, such as multi-layer perceptrons (MLP), to predict the attention scores over different LinearGNNs.

Robust Dimension Generalization with Entropy Normalization. While the distance features enable us to build a permutation-invariant attention module, they still rely on the assumption that the dimensions of feature- and label-spaces do not change across graphs. In practice, the dimensions of label spaces often vary a lot across datasets (e.g. 7 classes for Cora and 70 classes for FullCora). Consequently, the distance features have substantially varying scales (e.g. 7 versus 70) for different datasets, and the maximum and minimum distance become indistinguishable when the dimensions increase [5]. Both seriously hamper the generalization ability of the attention module trained on a single graph. While it is possible to offset the scale with a temperature hyperparameter specified for each dataset, this is contrary to our goal of graph foundation models. Instead, we would like a solution that automatically normalizes the distance features for any label dimension.

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:

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

¹It is important to distinguish between *domain symmetry* (in this case, permutation of the nodes of the graph) and *data symmetry* (permutation of the feature and label dimensions). Invariance to domain symmetry (node permutations) is provided by design in our LinearGNN.

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 $y_u^{(i)}$, which significantly reduces the gap between training and test features. We will verify the effectiveness of entropy normalization in Section 4.4.

3.3 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 $2.95\times$ 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 ($15.04\times$)
GraphAny	$O(\mathcal{E})$	$O(\mathcal{V}_L)$	$O(\mathcal{V})$	6.37 min ($2.95\times$)

4 Experiments

In this section, we evaluate the performance of GraphAny against both transductive and inductive methods on 31 node classification datasets (Section B). We visualize the attention of GraphAny on different datasets, shedding light on what inductive knowledge our model has learned (Section 4.3). To provide a comprehensive understanding of GraphAny, we further conduct ablation studies on the proposed inductive attention and entropy normalization (Section 4.4,4.5).

4.1 Experimental Setup

Datasets. We have compiled a diverse collection of 31 node classification datasets from three sources: PyG [12], DGL [43], and OGB [19]. 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. Detailed statistics for each dataset are provided in Appendix B.

Implementation Details. For GraphAny, we employ 5 LinearGNNs with different graph convolution operations: $F = X$ (Linear), $F = AX$ (LinearSGC1), $F = A^2X$ (LinearSGC2), $F = (I - A)X$ (LinearHGC1) and $F = (I - A)^2X$ (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, ensuring that the evaluations on these datasets are conducted in a fully inductive manner. More implementation details can be found at Appendix C. Code and data is available at this link².

Baselines. We compare GraphAny against conventional transductive methods, including MLP, GCN [21] and GAT [42]. These transductive models are trained separately for each dataset and can be considered as strong baselines for evaluating inductive models. Since there have been no inductive

²github.com/DeepGraphLearning/GraphAny

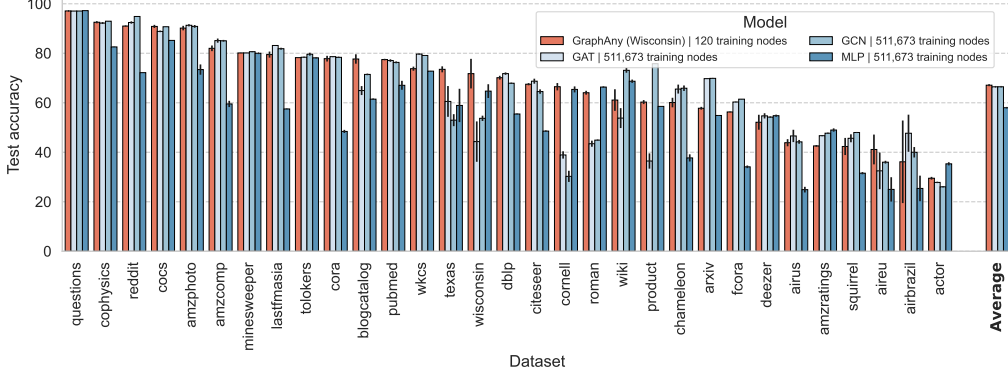


Figure 5: Inductive test accuracy (%) of GraphAny pre-trained using 120 labeled nodes of the Wisconsin dataset on 30 diverse graphs. Baseline methods are trained individually on each graph (511k labeled nodes in total). GraphAny is slightly better than the baselines in average performance.

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 \pm 0.63	79.21 \pm 1.07	55.50 \pm 0.23	61.06 \pm 0.08	57.09	57.20
	GCN	81.40 \pm 0.70	54.51 \pm 0.88	71.74 \pm 0.29	75.79 \pm 0.12	65.55	66.55
	GAT	81.70\pm1.43	52.94 \pm 3.10	73.65\pm0.11	79.45\pm0.59	65.31	67.03
Non-parametric	LabelProp	60.30 \pm 0.00	16.08 \pm 2.15	0.98 \pm 0.00	74.50 \pm 0.00	50.73 \pm 0.31	49.01 \pm 0.27
	Linear	52.80 \pm 0.00	80.00\pm2.15	46.79 \pm 0.00	42.10 \pm 0.00	57.91 \pm 0.43	57.59 \pm 0.42
	LinearSGC1	74.30 \pm 0.00	45.49 \pm 13.96	55.33 \pm 0.00	56.58 \pm 0.00	62.69 \pm 0.24	62.08 \pm 0.48
	LinearSGC2	78.20 \pm 0.00	57.64 \pm 1.07	59.58 \pm 0.00	62.92 \pm 0.00	64.38 \pm 0.48	64.41 \pm 0.39
	LinearHGC1	22.50 \pm 0.00	64.32 \pm 2.15	22.92 \pm 0.00	15.00 \pm 0.00	37.01 \pm 0.20	36.26 \pm 0.23
	LinearHGC2	23.80 \pm 0.00	56.08 \pm 4.29	20.65 \pm 0.00	13.39 \pm 0.00	35.62 \pm 0.68	34.70 \pm 0.55
Inductive (training set)	GraphAny (Cora)	80.18 \pm 0.13	61.18 \pm 5.08	58.62 \pm 0.05	61.60 \pm 0.10	67.24 \pm 0.23	67.00 \pm 0.14
	GraphAny (Wisconsin)	77.82 \pm 1.15	71.77 \pm 5.98	57.79 \pm 0.56	60.28 \pm 0.80	67.31 \pm 0.38	67.26 \pm 0.20
	GraphAny (Arxiv)	79.38 \pm 0.16	65.10 \pm 3.22	58.68 \pm 0.17	61.31 \pm 0.20	67.65 \pm 0.31	67.46 \pm 0.27
	GraphAny (Products)	79.36 \pm 0.23	65.89 \pm 2.23	58.58 \pm 0.11	61.19 \pm 0.23	67.66\pm0.39	67.48\pm0.33

models for node classification on any graph, we consider non-parametric methods as baselines, including label propagation (LabelProp) [51] and all 5 LinearGNNs used in GraphAny. Note that these methods perform inductive inference but do not transfer any knowledge across graphs.

4.2 Inductive Inference on Any Graph

Table 2 and Figure 5 show the results of GraphAny and baselines on 31 node classification datasets. We can see that our proposed LinearGNNs, with proper graph convolution operations, yield decent performance despite being non-parametric. Specifically, LinearSGC2 is only 2.1% lower than GCN in accuracy, which coincides with the observation that SGC has performance similar to that of GCN [44]. However, LinearSGC2 uses an analytical solution for inference, which is about 15 \times faster than training a GCN from scratch on each dataset (Table 1).

By combining multiple LinearGNNs with learned attention, GraphAny further improves over LinearSGC2 by 3.0% accuracy, outperforming all transductive models. This improvement mainly comes from inductive generalization, since GraphAny achieves the best performance on the 27 held-out datasets rather than the 4 datasets we use to train GraphAny. An intriguing finding is the lack of significant difference between GraphAny trained on small datasets (e.g. Cora and Wisconsin) and large datasets (e.g. Arxiv and Products). We conjecture the reason is that even small datasets contain sufficiently diverse node patterns, allowing GraphAny to learn an effective attention mechanism.

4.3 Visualization of Learned Attention

To understand how LinearGNNs are combined in GraphAny by the inductive attention, we visualize the attention weights of GraphAny (Wisconsin) and GraphAny (Arxiv) on all datasets, averaged across nodes. For reference, we also visualize the performance of each individual LinearGNN on all

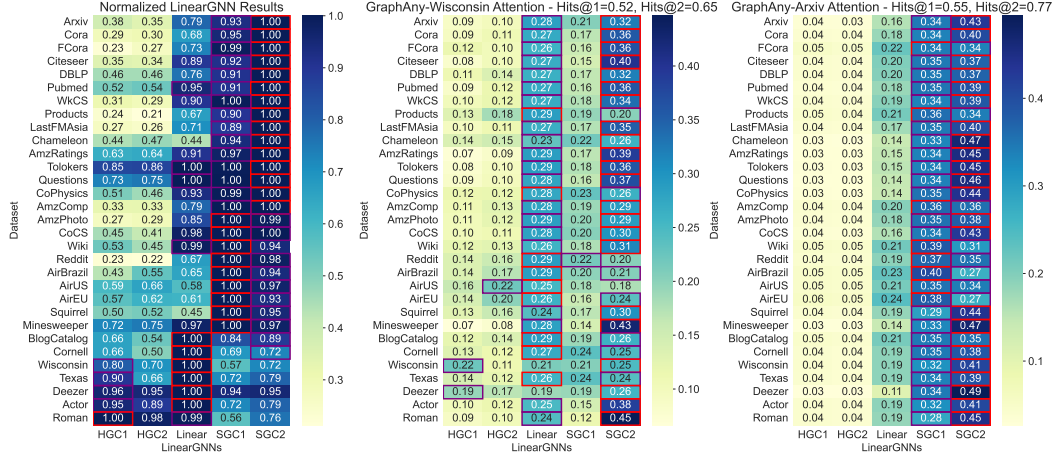


Figure 6: Normalized performance of LinearGNNs (left, best as 1) and attention weights of GraphAny trained on Wisconsin (middle) and Arxiv (right) respectively. The best and second best LinearGNN performance and attention weights for each dataset are highlighted with red and purple rectangles respectively.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

datasets. As shown in Figure 6, we can see that LinearSGC2 is optimal for half of the datasets, while the other half prefers LinearHGC1, Linear or LinearSGC1. In most cases, GraphAny successfully identifies the optimal LinearGNN within its top-2 attention weights, with Hits@2 being 0.65 and 0.77 for GraphAny trained on Wisconsin and Arxiv respectively.

Interestingly, there is a distinction between the attention distributions learned on the two datasets: Wisconsin leads a relatively balanced distribution of attention across 5 LinearGNNs, while Arxiv prefers a more focused distribution of attention, favoring low-pass filters like LinearSGC1 and LinearSGC2. This reflects the nature of these training sets: Wisconsin is a heterophilic dataset where all LinearGNNs are reasonably good, but Arxiv is a homophilic dataset where Linear, LinearSGC1 and LinearSGC2 have better performance (Table 2). Note that this does not necessarily mean GraphAny cannot attend to a LinearGNN that is not preferred by the training graph **at node level**, but the averaged attention **at graph level** does correlate with the distribution of the training graph.

4.4 Ablation Study on Entropy Normalization

One key component in GraphAny is the entropy normalization technique applied to distance features. To provide insights on how entropy normalization improves the dimension generalization ability of our method, we plot the feature distribution between the predictions of LinearSGC2 and other LinearGNNs on Cora and FullCora, which have similar semantics but different number of classes. As shown in Figure 7, there is a large difference in the scale of unnormalized feature distributions. By contrast, with entropy normalization, the feature distributions become very similar on these datasets, which facilitates inductive generalization across different datasets.

From the performance perspective, we observe a distinct trend in the generalization performance of normalized and unnormalized features. Figure 8 shows that unnormalized features, such as Euclidean distance and Jensen-Shannon divergence, provides a better test performance in the transductive setting. However, their test performance in the inductive setting diverges as training continues, indicating the model overfits the transductive information in the features, e.g. the scale of features. By contrast, entropy-normalized features achieve steady convergence in both transductive and inductive performance. Besides, the performance of entropy normalization is robust to different entropy values.

4.5 Ablation Study on Attention Parameterization

Our attention is parameterized as a function of distance features between LinearGNN predictions on each node (*node attention*). Here we consider two variants of attention parameterization: (1) *Graph attention* uses distance features averaged over all nodes in a training batch, which results in the same

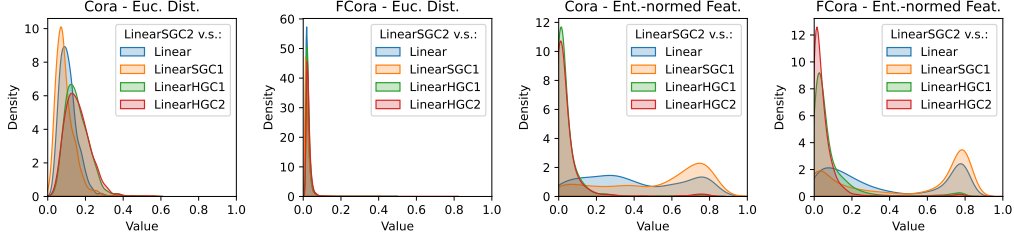


Figure 7: Cora (7 classes) vs Full Cora (70 classes). Left two plots: Without entropy normalization, there is a significant disparity in feature distributions between datasets. Right two plots: With entropy normalization applied, the disparity in feature distributions across datasets is largely reduced.

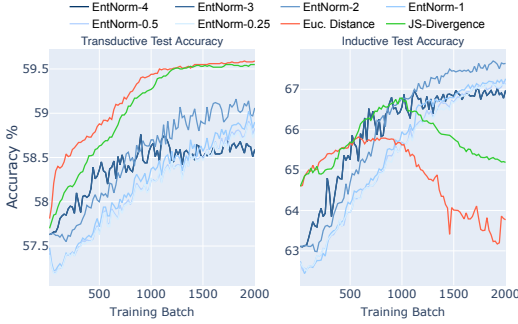


Figure 8: Performance of different distance features with and without entropy normalization.

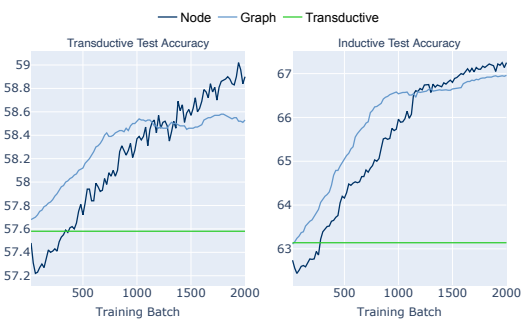


Figure 9: Performance of different attention parameterizations.

attention for all nodes in a training batch. (2) *Transductive attention* directly learns attention weights as parameters similar to transductive methods, and assumes they can transfer to new graphs.

Figure 9 plots the test performance curves for different attention parameterizations. We notice that transductive attention does not even learn anything useful, given its performance is worse than a single LinearSGC2 model (cf. Table 2). Comparing node attention and graph attention, we can see that node attention converges slower than graph attention, but results in better performance in both transductive and inductive settings. This suggests the effectiveness of learning fine-grained attention for each node to fuse multiple LinearGNNs in GraphAny.

5 Conclusion

Limitations and Future Work. Although GraphAny showcases the capacity of graph foundation models for node classification on any graph, there are several limitations and open questions. 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-parametric models have never been shown effective for relational graphs. We plan to address these open questions in our future work.

Societal Impact. GraphAny provides a efficient and effective tool for node classification on any graph. On the positive side, this largely reduces the carbon footprint of developing machine learning models on graphs. On a slightly negative side, our tool makes node classification more accessible and users can easily apply our tool to domains other than those studied in this paper.

Conclusion. This paper presents GraphAny, a foundation model for node classification on any graph with any feature and label spaces. GraphAny consists of two main components, LinearGNNs and an inductive attention module. LinearGNNs efficiently solve inductive inference on new graphs, while the inductive attention module learns an attention to combine multiple LinearGNN predictions. By training on a single dataset, GraphAny can effectively generalize to 30 new graphs and even surpass the average performance of transductive models supervisedly trained on each dataset.

References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Ravichandra Addanki, Peter W Battaglia, David Budden, Andreea Deac, Jonathan Godwin, Thomas Keck, Wai Lok Sibon Li, Alvaro Sanchez-Gonzalez, Jacklynn Stott, Shantanu Thakoor, et al. Large-scale graph representation learning with very deep gnns and self-supervision. *arXiv preprint arXiv:2107.09422*, 2021.
- [3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35: 23716–23736, 2022.
- [4] Ilyes Batatia, Philipp Benner, Yuan Chiang, Alin M. Elena, Dávid P. Kovács, Janosh Riebesell, Xavier R. Advincula, Mark Asta, Matthew Avaylon, William J. Baldwin, Fabian Berger, Noam Bernstein, Arghya Bhowmik, Samuel M. Blau, Vlad Cărare, James P. Darby, Sandip De, Flaviano Della Pia, Volker L. Deringer, Rokas Elijošius, Zakariya El-Machachi, Fabio Falcioni, Edwin Fako, Andrea C. Ferrari, Annalena Genreith-Schriever, Janine George, Rhys E. A. Goodall, Clare P. Grey, Petr Grigorev, Shuang Han, Will Handley, Hendrik H. Heenen, Kersti Hermansson, Christian Holm, Jad Jaafar, Stephan Hofmann, Konstantin S. Jakob, Hyunwook Jung, Venkat Kapil, Aaron D. Kaplan, Nima Karimitari, James R. Kermode, Namu Kroupa, Jolla Kullgren, Matthew C. Kuner, Domantas Kuryla, Guoda Liepuoniute, Johannes T. Margraf, Ioan-Bogdan Magdău, Angelos Michaelides, J. Harry Moore, Aakash A. Naik, Samuel P. Niblett, Sam Walton Norwood, Niamh O’Neill, Christoph Ortner, Kristin A. Persson, Karsten Reuter, Andrew S. Rosen, Lars L. Schaaf, Christoph Schran, Benjamin X. Shi, Eric Sivonxay, Tamás K. Stenczel, Viktor Svahn, Christopher Sutton, Thomas D. Swinburne, Jules Tilly, Cas van der Oord, Eszter Varga-Umbrich, Tejs Vegge, Martin Vondrák, Yangshuai Wang, William C. Witt, Fabian Zills, and Gábor Csányi. A foundation model for atomistic materials chemistry. *arXiv preprint arXiv:2401.00096*, 2024.
- [5] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is ”nearest neighbor” meaningful? In Catriel Beeri and Peter Buneman, editors, *Database Theory - ICDT ’99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999. doi: 10.1007/3-540-49257-7_15. URL https://doi.org/10.1007/3-540-49257-7_15.
- [6] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=r1ZdKJ-0W>.
- [7] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.
- [8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [9] Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. *arXiv preprint arXiv:2310.05845*, 2023.
- [10] Kaiwen Dong, Haitao Mao, Zhichun Guo, and Nitesh V. Chawla. Universal link predictor by in-context learning on graphs. *arXiv preprint arXiv:2402.07738*, 2024.
- [11] Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. Talk like a graph: Encoding graphs for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=IuXR1CCrSi>.
- [12] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

- [13] Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=jVEoydF019>.
- [14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [15] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.
- [16] Mengyue Hang, Jennifer Neville, and Bruno Ribeiro. A collective learning framework to boost gnn expressiveness for node classification. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 4040–4050. PMLR, 2021.
- [17] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.
- [18] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.
- [19] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs, 2021.
- [20] Hyosoon Jang, Seonghyun Park, Sangwoo Mo, and Sungsoo Ahn. Diffusion probabilistic models for structured node classification. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL <https://openreview.net/forum?id=CxUuCyMDU>.
- [21] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.
- [22] Kerstin Kläser, Błażej Banaszczyński, Samuel Maddrell-Mander, Callum McLean, Luis Müller, Ali Parviz, Shenyang Huang, and Andrew Fitzgibbon. MiniMol: A parameter-efficient foundation model for molecular learning. *arXiv preprint arXiv:2404.14986*, 2024.
- [23] Dávid Péter Kovács, J. Harry Moore, Nicholas J. Browning, Ilyes Batatia, Joshua T. Horton, Venkat Kapil, William C. Witt, Ioan-Bogdan Magdău, Daniel J. Cole, and Gábor Csányi. Mace-off23: Transferable machine learning force fields for organic molecules. *arXiv preprint arXiv:2312.15211*, 2023.
- [24] Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. One for all: Towards training one graph model for all classification tasks. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=4IT2pgc9v6>.
- [25] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Revisiting heterophily for graph neural networks. In *NeurIPS*, 2022.
- [26] Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Graph foundation models. *arXiv preprint arXiv:2402.02216*, 2024.
- [27] Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.
- [28] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.
- [29] Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. Let your graph do the talking: Encoding structured data for llms. *arXiv preprint arXiv:2402.05862*, 2024.
- [30] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at evaluation of gnns under heterophily: Are we really making progress? In *The Eleventh International Conference on Learning Representations*, 2023.

- [31] Meng Qu, Huiyu Cai, and Jian Tang. Neural structured prediction for inductive node classification. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=YWNAX0caEjI>.
- [32] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2017.
- [33] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021.
- [34] Ryoma Sato. Training-free graph neural networks and the power of labels as features. *arXiv preprint arXiv:2404.19288*, 2024.
- [35] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.
- [36] Nima Shoghi, Adeesh Kolluru, John R. Kitchin, Zachary Ward Ulissi, C. Lawrence Zitnick, and Brandon M Wood. From molecules to materials: Pre-training large generalizable models for atomic property prediction. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=PfPnugdxup>.
- [37] Maciej Sypetkowski, Frederik Wenkel, Farimah Poursafaei, Nia Dickson, Karush Suri, Philip Fradkin, and Dominique Beaini. On the scalability of gnns for molecular graphs. *arXiv preprint arXiv:2404.11568*, 2024.
- [38] Jiabin Tang, Yuhao Yang, Wei Wei, Lei Shi, Lixin Su, Suqi Cheng, Dawei Yin, and Chao Huang. Graphgpt: Graph instruction tuning for large language models. *arXiv preprint arXiv:2310.13023*, 2023.
- [39] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- [40] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [41] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL <http://jmlr.org/papers/v9/vandermaaten08a.html>.
- [42] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.
- [43] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2020.
- [44] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.
- [45] Han Yang, Chenxi Hu, Yichi Zhou, Xixian Liu, Yu Shi, Jielan Li, Guanzhi Li, Zekun Chen, Shuizhou Chen, Claudio Zeni, Matthew Horton, Robert Pinsler, Andrew Fowler, Daniel Zügner, Tian Xie, Jake Smith, Lixin Sun, Qian Wang, Lingyu Kong, Chang Liu, Hongxia Hao, and Ziheng Lu. Mattersim: A deep learning atomistic model across elements, temperatures and pressures. *arXiv preprint arXiv:2405.04967*, 2024.
- [46] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Sourav S Bhowmick, and Juncheng Liu. Pane: scalable and effective attributed network embedding. *The VLDB Journal*, 32(6): 1237–1262, 2023.

- [47] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 40–48. PMLR, 2016.
- [48] Duo Zhang, Xinzijian Liu, Xiangyu Zhang, Chengqian Zhang, Chun Cai, Hangrui Bi, Yiming Du, Xuejian Qin, Jiameng Huang, Bowen Li, Yifan Shan, Jinzhe Zeng, Yuzhi Zhang, Siyuan Liu, Yifan Li, Junhan Chang, Xinyan Wang, Shuo Zhou, Jianchuan Liu, Xiaoshan Luo, Zhenyu Wang, Wanrun Jiang, Jing Wu, Yudi Yang, Jiyuan Yang, Manyi Yang, Fu-Qiang Gong, Linshuang Zhang, Mengchao Shi, Fu-Zhi Dai, Darrin M. York, Shi Liu, Tong Zhu, Zhicheng Zhong, Jian Lv, Jun Cheng, Weile Jia, Mohan Chen, Guolin Ke, Weinan E, Linfeng Zhang, and Han Wang. Dpa-2: Towards a universal large atomic model for molecular and material simulation. *arXiv preprint arXiv:2312.15492*, 2023.
- [49] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 9061–9073, 2021.
- [50] Jianan Zhao, Le Zhuo, Yikang Shen, Meng Qu, Kai Liu, Michael Bronstein, Zhaocheng Zhu, and Jian Tang. Graphtext: Graph reasoning in text space. *arXiv preprint arXiv:2310.01089*, 2023.
- [51] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. *Technical Report CMU-CALD-02-107*, 2002.

A Proof of Feature and Label Permutation Invariance

A.1 Label-permutation Invariance

In this section, we show the label-permutation invariance of the distance of LinearGNN predictions. I.e. for any two LinearGNNs i and j , the (squared) distances between the permuted predictions, denoted as $\|\tilde{\mathbf{y}}_u^{(i)} - \tilde{\mathbf{y}}_u^{(j)}\|^2$ and the distances between the original predictions, i.e. $\|\hat{\mathbf{y}}_u^{(i)} - \hat{\mathbf{y}}_u^{(j)}\|^2$ are the same.

Consider permuting the label dimension, i.e. right multiplication with a permutation matrix $\mathbf{Q} \in \mathbb{R}^{c \times c}$. The permuted label logits $\tilde{\mathbf{Y}}$ can be expressed as:

$$\tilde{\mathbf{Y}} = \mathbf{F}\mathbf{F}_L^+ \mathbf{Y}_L \mathbf{Q}. \quad (11)$$

Given the linearity of matrix multiplication, we have:

$$\tilde{\mathbf{Y}} = \mathbf{F}\mathbf{F}_L^+ (\mathbf{Y}_L \mathbf{Q}) = (\mathbf{F}\mathbf{F}_L^+ \mathbf{Y}_L) \mathbf{Q} = \hat{\mathbf{Y}} \mathbf{Q}. \quad (12)$$

Thus, label-permutation *equivariance* is proved. As the distances between the permutation-equivariant distributions remain invariant, we have:

$$\begin{aligned} \|\tilde{\mathbf{y}}_u^{(i)} - \tilde{\mathbf{y}}_u^{(j)}\|^2 &= \hat{\mathbf{y}}_u^{(i)T} \mathbf{Q}^T \mathbf{Q} \hat{\mathbf{y}}_u^{(i)} + \hat{\mathbf{y}}_u^{(j)T} \mathbf{Q}^T \mathbf{Q} \hat{\mathbf{y}}_u^{(j)} - 2\hat{\mathbf{y}}_u^{(i)T} \mathbf{Q}^T \mathbf{Q} \hat{\mathbf{y}}_u^{(j)} \\ &= \hat{\mathbf{y}}_u^{(i)T} \hat{\mathbf{y}}_u^{(i)} + \hat{\mathbf{y}}_u^{(j)T} \hat{\mathbf{y}}_u^{(j)} - 2\hat{\mathbf{y}}_u^{(i)T} \hat{\mathbf{y}}_u^{(j)} \\ &= \|\hat{\mathbf{y}}_u^{(i)} - \hat{\mathbf{y}}_u^{(j)}\|^2. \end{aligned} \quad (13)$$

Hence, the distance-based feature is label permutation-invariant.

A.2 Feature-permutation Invariance

Now, let's consider permuting the feature dimension by right multiplication with a permutation matrix $\mathbf{P} \in \mathbb{R}^{d \times d}$, resulting in permuted features on labeled nodes denoted as $\mathbf{F}_L \mathbf{P}$. Since the permutation matrix \mathbf{P} is orthonormal, the pseudoinverse of $\mathbf{F}_L \mathbf{P}$ is:

$$(\mathbf{F}_L \mathbf{P})^+ = \mathbf{P}^\top \mathbf{F}_L^+. \quad (14)$$

Substituting it into the equation 4, the predictions using permuted feature are:

$$\mathbf{F} \mathbf{P} (\mathbf{F}_L \mathbf{P})^+ \mathbf{Y}_L = \mathbf{F} \mathbf{P} \mathbf{P}^\top \mathbf{F}_L^+ \mathbf{Y}_L = \mathbf{F} \mathbf{F}_L^+ \mathbf{Y}_L = \hat{\mathbf{Y}}. \quad (15)$$

Here, $\mathbf{P} \mathbf{P}^\top = \mathbf{I}$ where $\mathbf{I} \in \mathbb{R}^{d \times d}$ is the identity matrix, hence proving that the predictions with feature-permutation are identical to the original predictions $\hat{\mathbf{Y}}$. That is, the predictions are feature-permutation invariant.

B Datasets

We collect a diverse collection of 31 node classification datasets from three sources: PyG [12], DGL [43], and OGB [19]. We follow the default split if there is a given one, otherwise, we use the standard semi-supervised setting [21] where 20 nodes are randomly selected as training nodes for each label. The detailed dataset information is summarized in Table 3.

C Implementation Details

All experiments were conducted using five different random seeds: $\{0, 1, 2, 3, 4\}$. The best hyperparameters were selected based on the validation accuracy. The runtime measurements presented

Table 3: 31 datasets used in this paper. Of those, 20 are homophilic and 11 are heterophilic.

Dataset	Nodes	Edges	Feature Dim	Num classes	Num labeled nodes	Category	Source
Air Brazil	131	1074	131	4	80	Homophilic	[32]
Cornell	183	298	1703	5	87	Heterophilic	[28]
Texas	183	558	1703	5	87	Heterophilic	[28]
Wisconsin	251	515	1703	5	120	Heterophilic	[28]
Air EU	399	5995	399	4	80	Homophilic	[32]
Air US	1190	13599	1190	4	80	Homophilic	[32]
Chameleon	2277	36101	2325	5	1092	Heterophilic	[28]
Wiki	2405	17981	4973	17	340	Homophilic	[46]
Cora	2708	10556	1433	7	140	Homophilic	[47]
Citeseer	3327	9104	3703	6	120	Homophilic	[47]
BlogCatalog	5196	343486	8189	6	120	Homophilic	[46]
Squirrel	5201	217073	2089	5	2496	Heterophilic	[28]
Actor	7600	30019	932	5	3648	Heterophilic	[28]
LastFM Asia	7624	55612	128	18	360	Homophilic	[33]
AmzPhoto	7650	238162	745	8	160	Homophilic	[35]
Minesweeper	10000	78804	7	2	5000	Heterophilic	[30]
WikiCS	11701	431726	300	10	580	Homophilic	[27]
Tolokers	11758	1038000	10	2	5879	Heterophilic	[30]
AmzComp	13752	491722	767	10	200	Homophilic	[35]
DBLP	17716	105734	1639	4	80	Homophilic	[6]
CoCS	18333	163788	6805	15	300	Homophilic	[35]
Pubmed	19717	88648	500	3	60	Homophilic	[47]
FullCora	19793	126842	8710	70	1400	Homophilic	[6]
Roman Empire	22662	65854	300	18	11331	Heterophilic	[30]
Amazon Ratings	24492	186100	300	5	12246	Heterophilic	[30]
Deezer	28281	185504	128	2	40	Homophilic	[33]
CoPhysics	34493	495924	8415	5	100	Homophilic	[35]
Questions	48921	307080	301	2	24460	Heterophilic	[30]
Arxiv	169343	1166243	128	40	90941	Homophilic	[19]
Reddit	232965	114615892	602	41	153431	Homophilic	[15]
Product	2449029	123718280	100	47	196615	Homophilic	[19]

in Table 1 were performed on an NVIDIA Quadro RTX 8000 GPU with CUDA version 12.2, supported by an AMD EPYC 7502 32-Core Processor that features 64 cores and a maximum clock speed of 2.5 GHz. All GraphAny experiments can be conducted on a single GPU with 20GB GPU memory and 50GB CPU memory.

C.1 GraphAny Implementation

Table 4: Summary of hyperparameters of GraphAny on different datasets.

Dataset	# Batches	Learning Rate	Hidden Dimension	# MLP Layers	Entropy
Cora	500	0.0002	64	1	2
Wisconsin	1000	0.0002	32	2	1
Arxiv	1000	0.0002	128	2	1
Products	1000	0.0002	128	2	1

We optimize the attention module using standard cross-entropy loss on the labeled nodes. In each training batch (batch size set as 128 for all experiments), we randomly sample two disjoint sets of nodes from the labeled nodes \mathcal{V}_L : \mathcal{V}_{ref} and $\mathcal{V}_{\text{target}}$. \mathcal{V}_{ref} is used for performing inference using LinearGNNs, and $\mathcal{V}_{\text{target}}$ is utilized to compute the loss and update the attention module. This separation is intended to prevent the attention module from learning trivial solutions unless the number of labeled nodes is too low to allow for a meaningful split. Empirically, as the final attention weights of GraphAny mostly focus on Linear, LinearSGC1, and LinearSGC2 (Figure 6), we mask the attention for LinearHGC1 and LinearHGC2 to achieve faster convergence.

The hyperparameter search space for GraphAny is relatively small, we fixed the batch size as 128 and varied the number of training batches with options of 500, 1000, and 1500; explored hidden dimensions of 32, 64, and 128; tested configurations with 1, 2, and 3 MLP layers; and set the fixed entropy value H at 1 and 2. The optimal settings derived from this hyperparameter search space are detailed in Table 4.

C.2 Baseline Implementation

We utilize the Deep Graph Library (DGL) [43] implementations of GCN and GAT for our baseline models. To optimize their performance, we conducted a comprehensive hyperparameter tuning using a grid search strategy on each dataset. The search space included the following parameters: number of epochs fixed at 400; hidden dimensions explored were 64, 128, and 256; number of layers tested included 1, 2, and 3; and the learning rates considered were 0.0002, 0.0005, 0.001, and 0.002.

For label propagation, we use the DGL’s implementation and use grid search to find the best model with a search space defined as follows: number of propagation hops of 1, 2, and 3; α of 0.1, 0.3, 0.5, 0.7, and 0.9.

D Complete Results

Table 5 provides all results on 31 datasets for MLP, GCN, GAT baselines trained on each dataset from scratch and four GraphAny models trained only on one graph.

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