FULLY-INDUCTIVE NODE CLASSIFICATION ON ARBITRARY GRAPHS

Jianan Zhao[1,2,*], Zhaocheng Zhu[1,2,*], Mikhail Galkin[3†], Hesham Mostafa[4]
Michael Bronstein[5,6], Jian Tang[1,7,8]
[1]Mila - Québec AI Institute, [2]Université de Montréal, [3]Google Research [4]Intel Labs
[5]University of Oxford, [6]AITHYRA, [7]HEC Montréal, [8]CIFAR AI Chair

ABSTRACT

One fundamental challenge in graph machine learning is generalizing to new graphs. Many existing methods following the inductive setup can generalize to test graphs with new structures, but assuming the feature and label spaces remain the same as the training ones. This paper introduces the fully-inductive setup, where models should perform inference on *arbitrary test graphs with new structures, feature and label spaces*. We propose GraphAny as the first attempt at this challenging setup. GraphAny models inference on a new graph as an analytical solution to a LinearGNN, which can be naturally applied to graphs with any feature and label spaces. To further build a stronger model with learning capacity, we fuse multiple LinearGNN predictions with learned inductive attention scores. Specifically, the attention module is carefully parameterized as a function of the entropy-normalized distance features between pairs of LinearGNN predictions to ensure generalization to new graphs. Empirically, GraphAny trained on a single Wisconsin dataset with only 120 labeled nodes can generalize to 30 new graphs with an average accuracy of 67.26%, surpassing not only all inductive baselines, but also strong transductive methods trained separately on each of the 30 test graphs.

1 INTRODUCTION

One of the most important requirements for machine learning models is the ability to generalize to new data. Models with better generalization abilities are able to perform better on unseen data and tasks, which is a key property for foundation models (Achiam et al., 2023; Team et al., 2023; Touvron et al., 2023) that are designed to accomplish a wide range of downstream tasks. For graphs, generalization is challenging since different graphs usually have different structures and associated attributes. Ideally, graph machine learning models are expected to accommodate this difference and learn functions that are applicable to all graphs.

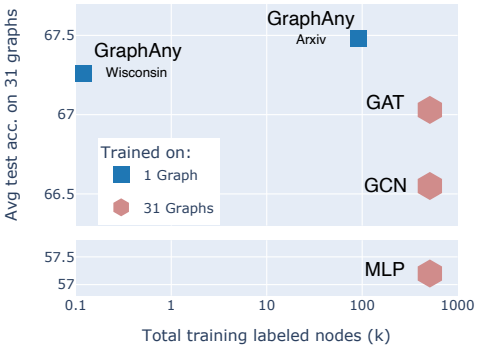


Figure 1: Average performance on 31 datasets. GraphAny is trained on a single dataset (Wisconsin or Arxiv) and performs inductive inference on any graph. The other methods have to be trained on each dataset.

Many previous works on graphs (Hamilton et al., 2017; Hang et al., 2021; Qu et al., 2022; Jang et al., 2023) consider generalization in the inductive setup, where models are supposed to perform inference on test graphs with new structures different from the training ones. However, these works rely on the assumption that the training and test graphs share the same feature and label spaces, which limits their applications to graphs in a fixed domain (e.g. social networks, citation networks). Ideally, we would like to have a model that generalizes to arbitrary graphs, involving new structures, new dimensions

[*]Equal contribution. Code release: `https://github.com/DeepGraphLearning/GraphAny`
[†]Work done while at Intel Labs

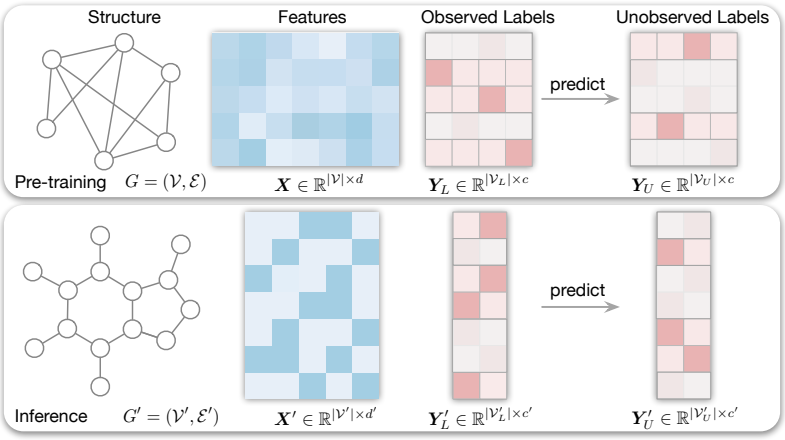


Figure 2: Fully-inductive node classification: Trained on a graph G, a fully-inductive model should generalize to any new graph G' with new feature and label spaces ***without additional training***.

and semantics for their feature and label spaces without additional training. We name this more general and practical setting as the *fully-inductive* setup (visualized in Figure 2).

The fully-inductive setup is particularly challenging for existing graph machine learning models for 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. Despite these challenges, it is always possible to cheat the fully-inductive setup by training a separate instance of existing models for each test dataset. We emphasize that this solution does not solve the fully-inductive setup. However, this cheating solution can be regarded as a strong baseline for fully-inductive models, since it additionally leverages back propagation on the labeled nodes and hyperparameter tuning on the validation data of the test datasets.

Our Contributions. We propose GraphAny to solve node classification in the fully-inductive setup. GraphAny consists of two components: *LinearGNNs* that perform inference on new feature and label spaces without training steps, and an *inductive attention* module 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 analytical 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 adaptively 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. By combining both modules, GraphAny learns to combine the LinearGNNs' predictions for each node based on their prediction distributions, which reflects statistics of its local structure, and generalizes to new graphs.

To summarize, the contribution of GraphAny are four folds:

- We introduce the fully-inductive setup for generalization across arbitrary graphs. Fully-inductive setup is more general and practical than the conventional inductive setup, allowing knowledge transfer across diverse domains, such as from knowledge graphs to e-commerce graphs.
- We devise LinearGNN, an analytical solution for node classification that enables efficient inductive inference on any new graph without the need for gradient descent.
- We identify two necessary properties for fully-inductive generalization: permutation invariance and dimensional robustness w.r.t. the feature and label spaces. Towards these goals, we design an inductive attention module that satisfies these properties and generalizes to new graphs.

- By combining LinearGNN and the inductive attention module, we present GraphAny, the first fully-inductive model for node classification. Across 31 datasets, GraphAny outperforms strong transductive baselines trained separately on each test dataset and achieves a 2.95× speedup.

2 Related Work

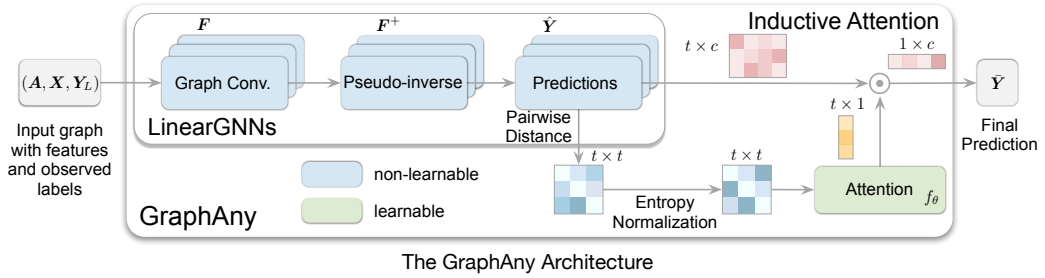
Inductive Node Classification. Depending on the test graphs which models generalize to, tasks on graphs can be categorized into *transductive* and *inductive* setups. In the transductive (semi-supervised) node classification setup (Kipf and Welling, 2017), test nodes belong to the same training graph the model was trained on. In the inductive setup (Hamilton et al., 2017), 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. (2021) introduced *Collective Learning* GNNs with iterative prediction refinement. Jang et al. (2023) applied a diffusion model for such prediction refinement. Structured Proxy Networks (Qu et al., 2022) 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 fully-inductive node classification, which generalizes to unseen graphs with arbitrary feature and label spaces.

Labels as Features. Addanki et al. (2021) demonstrated that node labels used as features yield noticeable improvements in the transductive node classification setup on MAG-240M in OGB-LSC (Hu et al., 2021a). Sato (2024) 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.

Connections to Graph Foundation Models. Graph foundation models (GFMs), which aim to develop a single model transferable to new graphs and diverse graph tasks, have garnered significant attention in the graph learning community. The core challenge in designing a GFM is achieving *generalization across graphs with varying input and output spaces*, which requires identifying an invariant feature space that transfers effectively across graphs (Mao et al., 2024). While fine-tuning a pre-trained model on a new graph is feasible (Zhu et al., 2024), we focus on the more challenging fully-inductive setting, where the model must generalize to unseen graphs *without* additional training.

In scenarios where *graphs share a common feature space*, such as in molecular learning (Kläser et al., 2024; Kovács et al., 2023; Zhang et al., 2023; Sypetkowski et al., 2024; Shoghi et al., 2024) and material design (Batatia et al., 2024), GNNs can be applied to the shared feature space of atoms, acting as GFMs. For *non-featurized graphs*, GNNs equipped with labeling tricks (Zhang et al., 2021) have shown promise as generalizable link predictors in homogeneous graphs (Dong et al., 2024) and multi-relational knowledge graphs (Galkin et al., 2024). Another line of research (Zhao et al., 2023; Tang et al., 2023; Chai et al., 2023; Fatemi et al., 2024; Perozzi et al., 2024; Liu et al., 2024) focuses on *text-attributed graphs* (Yan et al., 2023), where features are represented as, or converted into, text. Here, large language models (LLMs) serve as universal featurizers by verbalizing the (sub)graph structure and the associated task in natural language. However, it is unclear whether the sequential nature of LLMs is suitable for graphs with permutation invariance.

In summary, while existing GFMs have shown promising results, their generalization capabilities often rely on strong assumptions over the training and test graphs, particularly the presence of a shared input space. In contrast, GraphAny is the first attempt to the more general and challenging problem of generalizing across *arbitrary graphs*. This broader scope opens up new possibilities for transferring knowledge between different graph domains, such as from knowledge graphs to e-commerce graphs. Additionally, we believe our proposed model, along with essential generalization properties like permutation invariance and dimensional robustness, provide a solid foundation for future research on powerful and versatile GFMs.



The GraphAny Architecture

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.

3 GraphAny: Fully-inductive Node Classification on Any Graph

Our goal is to devise a fully-inductive 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. Each LinearGNN provides 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.

Formally, in a semi-supervised node classification task, we are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, typically represented as an adjacency matrix $\boldsymbol{A}$, and node features $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, a set of labeled nodes $\mathcal{V}_L$ and their labels $\boldsymbol{Y}_L \in \mathbb{R}^{|\mathcal{V}_L| \times c}$, where c is the number of unique label classes. The goal of node classification is to predict the labels $\hat{\boldsymbol{Y}}$ 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 (Kipf and Welling, 2017), GAT (Velickovic et al., 2018)) 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 some forms of re-training or fine-tuning. Conversely, a fully-inductive model is expected to predict the labels $\hat{\boldsymbol{Y}}$ for any graph without expensive gradient steps. Furthermore, when a new graph 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 Inductive Inference with LinearGNNs

A key idea of this paper is to use simple GNN models whose parameters can be expressed analytically. Following existing works that simplifies GNN (Wu et al., 2019; Zhu and Koniusz, 2021; Yoo et al., 2023) by removing non-linearity, we leverage graph convolutions 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 most existing GNNs are trained to minimize a cross-entropy loss on node classification, which does not have analytical solution and requires gradient descent to learn the weights. Alternatively, we propose to use a mean-squared error loss for optimizing the weights:

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

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 an analytical solution for the 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:

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

We term this architecture *LinearGNN*, as it approximates the prediction of linear GNNs (like SGC (Wu et al., 2019)) with a single forward pass. Its main advantage is that it requires no training, which makes it significantly more computationally efficient (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 do not 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 LinearGNN Predictions

Although LinearGNNs provide basic solutions to inductive inference on new graphs, the learned weights are still *transductive*, i.e. specific to the training graph, and do not capture transferable knowledge that can be applied to unseen graphs. Besides, our experiments (Figure 7) suggest that different graphs may require LinearGNNs with convolution operations. Hence, a natural way to incorporate fully-inductive learning is to add an *inductive attention module* over a set of multiple LinearGNNs. Let $\alpha_u^{(1)}, \alpha_u^{(2)}, ..., \alpha_u^{(t)}$ denote the node-level attention over t LinearGNNs. We generate the final prediction as a combination of all LinearGNN predictions:

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

While there are various ways to parameterize this attention module, finding an inductive solution is non-trivial since neural networks can easily fit the information specific to the training graph. We notice a necessary property for fully-inductive functions 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.

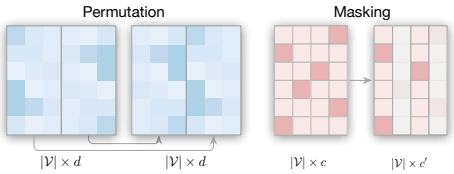


Figure 4: Transformations on graph features and labels: permutation (left), masking (right).

Permutation-Invariant Attention with Distance Features. We would like to design an attention module that is permutation invariant (Bronstein et al., 2021) 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 unpermutated 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 $\boldsymbol{P} \in \mathbb{R}^{d \times d}$ 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:

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

[1] It 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.

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 t 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 distances:

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

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.

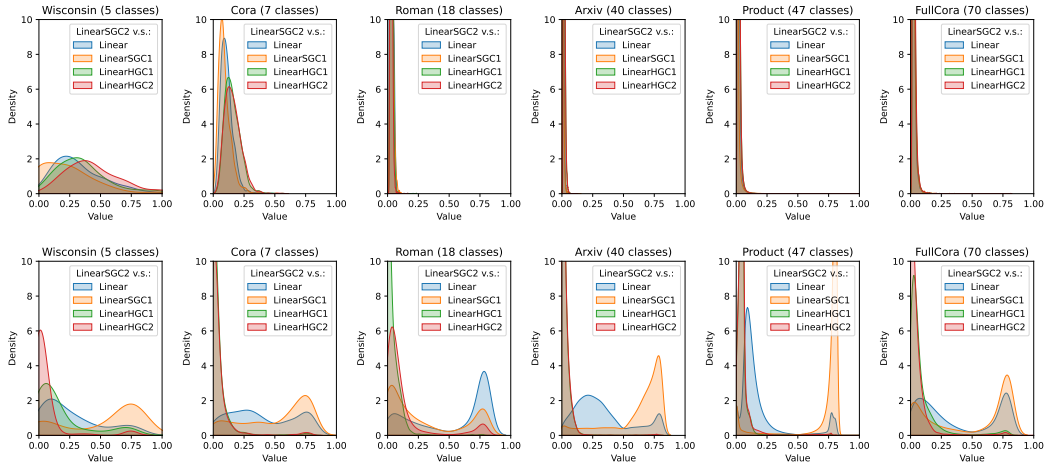


Figure 5: Comparison of Euclidean distances (the first row) and entropy-normalized (the second row) features between five channels: $\boldsymbol{F} = \boldsymbol{X}$ (Linear), $\boldsymbol{F} = \bar{\boldsymbol{A}}\boldsymbol{X}$ (LinearSGC1), $\boldsymbol{F} = \bar{\boldsymbol{A}}^2\boldsymbol{X}$ (LinearSGC2), $\boldsymbol{F} = (\boldsymbol{I} - \bar{\boldsymbol{A}})\boldsymbol{X}$ (LinearHGC1) and $\boldsymbol{F} = (\boldsymbol{I} - \bar{\boldsymbol{A}})^2\boldsymbol{X}$ (LinearHGC2) with $\bar{\boldsymbol{A}}$ denoting the row normalized adjaceny matrix. Entropy-normalized features are on the same scale and exhibit transferrable patterns across datasets.

Robust Dimension Generalization with Entropy Normalization. While the distance features for inductive attention ensure a permutation-invariant attention module, the distance features are known to suffer from the curse of dimensionality, where distances between vectors with larger dimensions have smaller scales (Beyer et al., 1999). This will hamper the generalization performance when the dimensions of label spaces vary across training and inference graphs (e.g. training on 7 classes for Cora and inference on 70 classes for FullCora). Empirically, as shown in the first row of Figure 5, the scale of Euclidean distance distributions decreases drastically when the number of classes increases, hampering the generalization ability of the attention module trained on a single graph.

A naive approach is to normalize the distance-feature distributions using a hyperparameter (e.g., temperature). However, due to the varying neighbor patterns across graphs (or even nodes (Luan et al., 2022)), a single hyperparameter may not be suitable for all nodes and graphs. Instead, we propose an adaptive solution that normalizes distance features to a consistent scale. To achieve this, we employ *entropy normalization*, a technique commonly used in manifold learning (Hinton and Roweis, 2002; van der Maaten and Hinton, 2008) to adaptively determine the similarity features. For node u, the asymmetric similarity feature between LinearGNN predictions i and j is defined as:

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

where $\sigma_u^{(i)}$ is the standard deviation of an isotropic multivariate Gaussian, determined by matching the entropy of distance distributions $P_u^{(i)} = \{p_u(j \mid i) \mid j \in [1, t]\}$ to a fixed hyperparameter H. Since the similarity features are derived from 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 considered similar to $\hat{\boldsymbol{y}}_u^{(i)}$, significantly reducing the gap between training and test features. As shown in the second row of Figure 5, entropy-normalized feature distributions are on consistent scales across datasets. Additionally, we observe that different types of homophilic graphs (e.g., the citation graph Cora and the e-commerce graph Product) exhibit similar entropy-normed features. We will further verify the effectiveness of entropy normalization empirically in Section 4.4.

3.3 Efficient Training and Fully-inductive Inference

In this section, we summarize how GraphAny utilizes the techniques introduced in the previous section and derive an efficient fully-inductive node classification model. As shown in Figure 3, given any graph, GraphAny first utilizes t LinearGNNs to provide basic predictions with different channels. Then, entropy-normalized features are computed based on the distances between these predictions, leading to $t(t-1)$-dimensional features. Further, an inductive attention module f_θ (e.g. MLP) is used to compute the attention scores for fusing different predictions into a final prediction (Eq. 5). Since the only trainable module of GraphAny lies in the inductive attention module $f_\theta : \mathbb{R}^{t(t-1)} \to \mathbb{R}^t$, which is *independent* to feature dimension d and label dimension c, GraphAny enjoys fully-inductive inference on any graph with arbitrary feature and label dimensions.

One advantage of GraphAny is that it is more efficient than conventional graph neural networks (e.g. GCN (Kipf and Welling, 2017), 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 the optimization complexity to $O(|\mathcal{V}_L|)$ compared with the complexity of $O(|\mathcal{E}|)$ for standard GNNs. Second, once trained, GraphAny is ready to generalize to arbitrary graphs, eliminating the need for gradient descent on test graphs.

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 the optimized DGL's (Wang et al., 2020) GCN implementation 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. 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	$O(\lvert\mathcal{E}\rvert)$	$O(\lvert\mathcal{E}\rvert)$	18.80 min
LinearGNN	$O(\lvert\mathcal{E}\rvert)$	$O(\lvert\mathcal{V}_L\rvert)$	$O(\lvert\mathcal{V}_U\rvert)$	1.25 min (15.04×)
GraphAny	$O(\lvert\mathcal{E}\rvert)$	$O(\lvert\mathcal{V}_L\rvert)$	$O(\lvert\mathcal{V}_U\rvert)$	6.37 min (2.95×)

4 Experiments

In this section, we evaluate the performance of GraphAny against both transductive and inductive methods on 31 node classification datasets (details in Appendix 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 the proposed techniques of GraphAny, we further conduct ablation studies on the entropy-normalized feature and attention parameterization in Section 4.4. More training and implementation details are provided in Appendix C.

4.1 Experimental Setup

Datasets. We have compiled a diverse collection of 31 node classification datasets from three sources: PyG (Fey and Lenssen, 2019), DGL (Wang et al., 2020), and OGB (Hu et al., 2021b). 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.

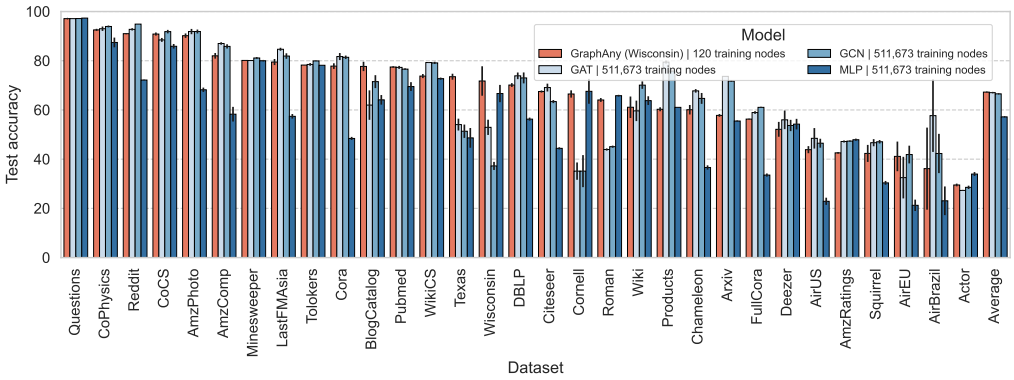


Figure 6: Inductive test accuracy (%) of GraphAny pre-trained using 120 labeled nodes of the Wisconsin dataset on 30 diverse graphs. Baseline methods are trained individually on each graph (511k labeled nodes in total). GraphAny is slightly better than the baselines in average performance.

Implementation Details. For GraphAny, we employ 5 LinearGNNs with different graph convolution operations: $\boldsymbol{F} = \boldsymbol{X}$ (Linear), $\boldsymbol{F} = \bar{\boldsymbol{A}}\boldsymbol{X}$ (LinearSGC1), $\boldsymbol{F} = \bar{\boldsymbol{A}}^2\boldsymbol{X}$ (LinearSGC2), $\boldsymbol{F} = (\boldsymbol{I} - \bar{\boldsymbol{A}})\boldsymbol{X}$ (LinearHGC1) and $\boldsymbol{F} = (\boldsymbol{I} - \bar{\boldsymbol{A}})^2\boldsymbol{X}$ (LinearHGC2) with $\bar{\boldsymbol{A}}$ denoting the row normalized adjaceny matrix, 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.

Baselines. As there are no existing fully-inductive node classification baselines, we include non-parametric methods (models without learnable parameters) like label propagation (Zhu and Ghahramani, 2002) and the five LinearGNNs used in GraphAny. While these methods perform inductive inference, they do not transfer knowledge across graphs. Additionally, we compare GraphAny with transductive models, including MLP, GCN (Kipf and Welling, 2017), and GAT (Velickovic et al., 2018). These models are trained separately on each dataset and serve as strong baselines for inductive models, as they benefit from backpropagation on labeled nodes and hyperparameter tuning on validation sets of the test dataset.

4.2 Performance of Inductive Node Classification

Table 2 presents the results of GraphAny and various baselines on 31 node classification datasets (complete results for each dataset are provided in Appendix D). Our proposed LinearGNNs, despite being non-parametric, demonstrate competitive performance. Notably, LinearSGC2, a linear model with a two-hop graph convolution layer, achieves only 2.1% lower accuracy than GCN, which aligns with previous findings that SGC performs comparably to GCN (Wu et al., 2019). Moreover, LinearSGC2 leverages an analytical solution for inference, making it approximately 15× faster than training a GCN from scratch on each dataset (see Table 1). Additionally, we observe that the optimal LinearGNN model differs across datasets, highlighting that no single graph convolution kernel is universally effective for all graphs.

As for GraphAny, which is trained on just 1 of the 31 graphs, it significantly outperforms LinearGNNs and even slightly surpasses transductive baselines that are individually trained on all 31 graphs. This improvement is primarily driven by inductive generalization, as GraphAny achieves its strongest performance on the 27 held-out (fully-inductive) datasets rather than the 4 training (transductive) datasets. A closer examination of Figure 6 shows that GraphAny performs well on both homophilic and heterophilic graphs in an inductive manner. We attribute this to the inductive attention module, which adaptively fuses predictions from different graph convolution kernels for each node. Interestingly, we also observe minimal performance differences between GraphAny when trained on small datasets (e.g., Cora and Wisconsin) and large datasets (e.g., Arxiv and Products). We hypothesize that even small datasets contain sufficiently diverse local node patterns (e.g., homophily

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	66.67±3.51	55.50±0.23	61.06±0.08	57.09	57.20
	GCN	81.40±0.70	37.25±1.64	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**

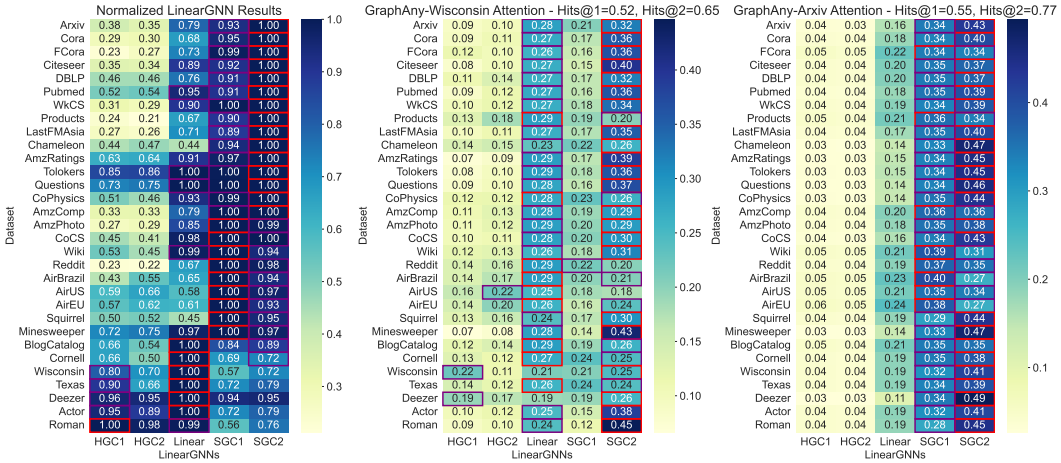


Figure 7: Normalized performance of LinearGNNs (left; best as 1.00) and attention weights of GraphAny trained on Wisconsin (middle) and Arxiv (right) respectively. The best and second best LinearGNN performance and attention weights for each dataset are highlighted with red and purple rectangles respectively. The learned inductive attention of GraphAny successfully identifies the best-performing LinearGNN for most datasets.

and heterophily) (Luan et al., 2022), enabling GraphAny to learn an effective node-level attention mechanism that generalizes well with a limited number of nodes (e.g., 120). A detailed analysis of the attention module is provided in Section 4.3.

4.3 Visualization of the Inductive 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 datasets. As shown in Figure 7, we can see that half of the datasets are homophilic with LinearSGC2 being the optimal LinearGNN, 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.

We hypothesis that this amazing inductive performance comes from the inductive entropy-normed distance feature we derived, where homophilic and heterophilic graphs share different patterns (see the second row in Figure 5). Interestingly, there is a distinction between the attention distributions when training on different datasets: GraphAny-Wisconsin leads a relatively balanced distribution of attention across 5 LinearGNNs, while Graph-Arxiv prefers a more focused distribution of attention, favoring low-pass filters like LinearSGC1 and LinearSGC2. This reflects the nature of these training sets: As shown in the left part of Figure 7, all LinearGNN channels in Wisconsin are reasonably good,

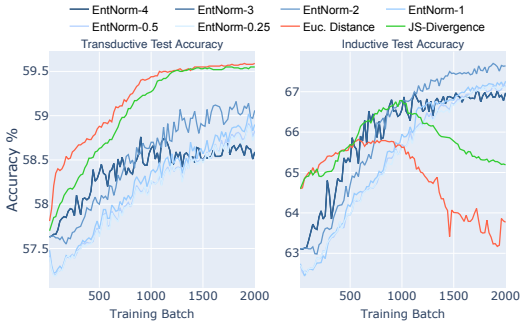


Figure 8: Performance of different distance features with and without entropy normalization.

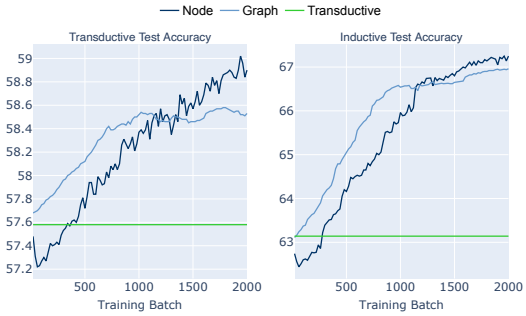


Figure 9: Performance of different attention parameterizations.

indicating diverse message-passing pattern exists, but Arxiv is a homophilic dataset where Linear, LinearSGC1 and LinearSGC2 have better performance (Table 2). These different message-passing patterns are learned and used by GraphAny to generate inductive attention scores.

4.4 Ablation Studies

Entropy Normalization. A key component of GraphAny is the entropy normalization technique applied to distance features. As discussed in Figure 5, entropy normalization ensures that the generated features are of a consistent scale. Here, we further evaluate entropy normalization from a performance perspective. Figure 8 demonstrates that unnormalized features, such as Euclidean distance and Jensen-Shannon divergence, yield better test performance in the transductive setting. However, their performance in the inductive setting decreases as training progresses, indicating that the model overfits to transductive information in the features, such as feature scale. In contrast, distance features normalized to the same entropy H (denoted as EntNorm-H in the figure) achieve stable convergence in both transductive and inductive settings. Additionally, entropy normalization shows robustness to the choice of the hyperparameter entropy value H, with different selections resulting in similar convergence rates and performance.

Attention Parameterization. It is known that the optimal message-passing patterns vary for different graphs (Zhu et al., 2020) or even different nodes (Luan et al., 2022). Therefore, GraphAny utilizes a *node-level attention* to adaptively combine different LinearGNN predictions. Here we consider two variants of attention parameterization: (1) *Graph-level attention* uses distance features averaged over all nodes in a training batch, which results in the same attention for all nodes in a training batch, losing the personalization for each node. (2) *Transductive attention* directly parameterizes attention weights as a t-dimensional vector and assumes they can transfer to new graphs. Figure 9 plots the transductive and inductive 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 (see Table 2). Comparing node-level attention and graph-level attention, we can see that node-level attention converges slower than graph-level attention, but results in better performance in both transductive and inductive settings. This suggests the effectiveness of learning fine-grained attention based on the local information of each node.

5 Conclusion

In this paper, we propose GraphAny, the first fully-inductive node classification model capable of performing inference on any graph with arbitrary feature or label space. GraphAny is composed of two core components: LinearGNNs and an inductive attention module. LinearGNNs enable efficient inductive inference on unseen graphs, while the inductive attention module learns to adaptively aggregate predictions from multiple LinearGNNs. Trained on a single graph, GraphAny demonstrates strong generalization to 30 new graphs, even surpassing the average performance of transductive models that are trained separately on each dataset.

ACKNOLEDGEMENT

The authors would like to thank Dinghuai Zhang for his helpful discussions and comments. This project is supported by the Intel-MILA partnership program, the Natural Sciences and Engineering Research Council (NSERC) Discovery Grant, the Canada CIFAR AI Chair Program, collaboration grants between Microsoft Research and Mila, Tencent AI Lab Rhino-Bird Gift Fund and a NRC Collaborative R&D Project (AI4DCORE-06). This project was also partially funded by IVADO Fundamental Research Project grant PRF-2019-3583139727. M.B. is partially supported by the EPSRC Turing AI World-Leading Research Fellowship No. EP/X040062/1 and EPSRC AI Hub No. EP/Y028872/1. The computation resource of this project is supported by Mila, Calcul Québec and the Digital Research Alliance of Canada.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

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.

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, Edvin 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.

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.

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

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.

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.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

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.

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.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

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.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

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.

Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.

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*, 2021a.

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, 2021b.

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.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Kerstin Kläser, Błażej Banaszewski, 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.

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.

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.

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.

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.

Péter Mernyei and Cătălina Cangea. Wiki-cs: A wikipedia-based benchmark for graph neural networks. *arXiv preprint arXiv:2007.02901*, 2020.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *ICLR*, 2020.

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.

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.

Meng Qu, Huiyu Cai, and Jian Tang. Neural structured prediction for inductive node classification. In *International Conference on Learning Representations*, 2022.

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.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale Attributed Node Embedding. *Journal of Complex Networks*, 9(2), 2021.

Ryoma Sato. Training-free graph neural networks and the power of labels as features. *arXiv preprint arXiv:2404.19288*, 2024.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

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.

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.

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.

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.

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.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

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.

Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Hao Yan, Chaozhuo Li, Ruosong Long, Chao Yan, Jianan Zhao, Wenwen Zhuang, Jun Yin, Peiyan Zhang, Weihao Han, Hao Sun, Weiwei Deng, Qi Zhang, Lichao Sun, Xing Xie, and Senzhang Wang. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 17238–17264. Curran Associates, Inc., 2023.

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.

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.

Jaemin Yoo, Meng-Chieh Lee, Shubhranshu Shekhar, and Christos Faloutsos. Less is more: Slimg for accurate, robust, and interpretable graph mining. In *KDD*, pages 3128–3139. ACM, 2023.

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.

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.

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.

Hao Zhu and Piotr Koniusz. Simple spectral graph convolution. In *ICLR*. OpenReview.net, 2021.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *NIPS*, 2020.

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. *Technical Report CMU-CALD-02-107*, 2002.

Yun Zhu, Yaoke Wang, Haizhou Shi, Zhenshuo Zhang, Dian Jiao, and Siliang Tang. Graphcontrol: Adding conditional control to universal graph pre-trained models for graph domain transfer learning. In *WWW*, 2024.

A Proof of Feature and Label Permutation Invariance

In this section, we prove that the proposed distance-based features have the desiring property of feature- and label-permutation invariance.

A.1 Label-permutation Invariance

Here, 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 $\|\tilde{\boldsymbol{y}}_u^{(i)} - \tilde{\boldsymbol{y}}_u^{(j)})\|^2$ and the distances between the original predictions, i.e. $\|\hat{\boldsymbol{y}}_u^{(i)} - \hat{\boldsymbol{y}}_u^{(j)}\|^2$ are the same.

Consider permuting the label dimension, i.e. right multiplicate a permutation matrix $\boldsymbol{Q} \in \mathbb{R}^{c\times c}$ to the LinearGNN prediction $\hat{\boldsymbol{Y}} = \boldsymbol{F}\boldsymbol{F}_L^+\boldsymbol{Y}_L$. (Eq 4). The permutated label logits $\tilde{\boldsymbol{Y}}$ can be expressed as:

$$\tilde{\boldsymbol{Y}} = \boldsymbol{F}\boldsymbol{F}_L^+\boldsymbol{Y}_L\boldsymbol{Q}. \tag{11}$$

Given the linearity of matrix multiplication, we have:

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

Thus, label-permutation *equivariance* is proved. Since the permutation matrix $\mathbf{Q}$ is orthonormal (i.e. $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$), pairwise distances between permuted logits remain the same as those between the original logits.

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

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 $\boldsymbol{P} \in \mathbb{R}^{d\times d}$, resulting in permuted features on labeled nodes denoted as $\boldsymbol{F}_L\boldsymbol{P}$. Since the permutation matrix $\boldsymbol{P}$ is orthonormal, the pseudoinverse of $\boldsymbol{F}_L\boldsymbol{P}$ is:

$$(\boldsymbol{F}_L\boldsymbol{P})^+ = \boldsymbol{P}^\top\boldsymbol{F}_L^+. \tag{14}$$

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

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

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

B Datasets

We collect a diverse collection of 31 node classification datasets from three sources: PyG (Fey and Lenssen, 2019), DGL (Wang et al., 2020), and OGB (Hu et al., 2021b). We follow the default split if there is a given one, otherwise, we use the standard semi-supervised setting (Kipf and Welling, 2017) where 20 nodes are randomly selected as training nodes for each label. The detailed dataset information is summarized in Table 3.

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

Dataset	#Nodes	#Edges	#Feature	#Classes	#Labeled Nodes	Train/Val/Test Ratios (%)	Category	Source
Air Brazil	131	1074	131	4	80	61.1/19.1/19.8	Homophilic	Ribeiro et al. (2017)
Cornell	183	554	1703	5	87	47.5/32.2/20.2	Heterophilic	Pei et al. (2020)
Texas	183	558	1703	5	87	47.5/31.7/20.2	Heterophilic	Pei et al. (2020)
Wisconsin	251	900	1703	5	120	47.8/31.9/20.3	Heterophilic	Pei et al. (2020)
Air EU	399	5995	399	4	80	20.1/39.8/40.1	Homophilic	Ribeiro et al. (2017)
Air US	1190	13599	1190	4	80	6.7/46.6/46.6	Homophilic	Ribeiro et al. (2017)
Chameleon	2277	36101	2325	5	1092	48.0/32.0/20.0	Heterophilic	Pei et al. (2020)
Wiki	2405	17981	4973	17	340	14.1/42.9/43.0	Homophilic	Yang et al. (2023)
Cora	2708	10556	1433	7	140	5.2/18.5/36.9	Homophilic	Yang et al. (2016)
Citeseer	3327	9104	3703	6	120	3.6/15.0/30.1	Homophilic	Yang et al. (2016)
BlogCatalog	5196	343486	8189	6	120	2.3/48.8/48.8	Homophilic	Yang et al. (2023)
Squirrel	5201	217073	2089	5	2496	48.0/32.0/20.0	Heterophilic	Pei et al. (2020)
Actor	7600	30019	932	5	3648	48.0/32.0/20.0	Heterophilic	Pei et al. (2020)
LastFM Asia	7624	55612	128	18	360	4.7/47.6/47.6	Homophilic	Rozemberczki et al. (2021)
AmzPhoto	7650	238162	745	8	160	2.1/49.0/49.0	Homophilic	Shchur et al. (2018)
Minesweeper	10000	78804	7	2	5000	50.0/25.0/25.0	Heterophilic	Platonov et al. (2023)
WikiCS	11701	431206	300	10	580	5.0/15.1/49.9	Homophilic	Mernyei and Cangea (2020)
Tolokers	11758	1038000	10	2	5879	50.0/25.0/25.0	Heterophilic	Platonov et al. (2023)
AmzComp	13752	491722	767	10	200	1.5/49.3/49.3	Homophilic	Shchur et al. (2018)
DBLP	17716	105734	1639	4	80	0.5/49.8/49.8	Homophilic	Bojchevski and Günnemann (2018)
CoCS	18333	163788	6805	15	300	1.6/49.2/49.2	Homophilic	Shchur et al. (2018)
Pubmed	19717	88648	500	3	60	0.3/2.5/5.1	Homophilic	Yang et al. (2016)
FullCora	19793	126842	8710	70	1400	7.1/46.5/46.5	Homophilic	Bojchevski and Günnemann (2018)
Roman Empire	22662	65854	300	18	11331	50.0/25.0/25.0	Heterophilic	Platonov et al. (2023)
Amazon Ratings	24492	186100	300	5	12246	50.0/25.0/25.0	Heterophilic	Platonov et al. (2023)
Deezer	28281	185504	128	2	40	0.1/49.9/49.9	Homophilic	Rozemberczki et al. (2021)
CoPhysics	34493	495924	8415	5	100	0.3/49.9/49.9	Homophilic	Shchur et al. (2018)
Questions	48921	307080	301	2	24460	50.0/25.0/25.0	Heterophilic	Platonov et al. (2023)
Arxiv	169343	1166243	128	40	90941	53.7/17.6/28.7	Homophilic	Hu et al. (2021b)
Reddit	232965	114615892	602	41	153431	65.9/10.2/23.9	Homophilic	Hamilton et al. (2017)
Product	2449029	123718280	100	47	196615	8.0/1.6/90.4	Homophilic	Hu et al. (2021b)

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 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 $\mathcal{V}_L$: $\mathcal{V}_{\text{ref}}$ and $\mathcal{V}_{\text{target}}$. $\mathcal{V}_{\text{ref}}$ 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 7), 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) (Wang et al., 2020) 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	33.95±0.80	28.55±0.68	27.30±0.22	28.99±0.61	28.60±0.21	29.51±0.55	27.91±0.16
AirBrazil	23.08±5.83	42.31±7.98	57.69±14.75	34.61±16.54	34.61±16.09	36.15±16.68	33.07±16.68
AirEU	21.25±2.31	41.88±3.60	32.50±8.45	41.75±6.84	41.50±6.50	41.13±6.02	40.50±7.01
AirUS	22.88±1.46	46.49±1.81	48.47±4.17	43.57±2.07	43.64±1.83	43.86±1.44	43.46±1.45
AmzComp	58.28±2.98	85.83±0.86	87.01±0.50	82.90±1.25	83.04±1.24	82.00±1.14	82.99±1.22
AmzPhoto	68.20±0.88	91.88±0.79	91.86±1.07	90.64±0.82	90.60±0.82	90.18±0.91	90.14±0.93
AmzRatings	47.90±0.45	47.35±0.26	47.18±0.42	42.70±0.10	42.74±0.12	42.57±0.34	42.84±0.04
BlogCatalog	64.11±1.95	71.51±2.62	61.98±5.99	74.73±3.19	73.63±2.95	77.69±1.90	72.52±3.22
Chameleon	36.62±0.87	64.69±2.21	67.76±0.72	62.59±0.87	62.59±0.86	60.09±1.93	61.49±1.88
Citeseer	44.40±0.44	63.40±0.63	69.10±1.59	67.94±0.29	68.34±0.23	67.50±0.44	68.90±0.07
CoCS	85.88±0.93	91.83±0.71	88.47±0.79	90.46±0.54	90.45±0.59	90.85±0.63	90.47±0.63
CoPhysics	87.43±1.98	93.93±0.37	93.01±0.89	92.66±0.52	92.69±0.52	92.54±0.43	92.70±0.54
Cora	48.42±0.63	81.40±0.70	81.70±1.43	79.36±0.23	79.38±0.16	77.82±1.15	80.18±0.13
Cornell	67.57±5.06	35.14±6.51	35.14±3.52	64.86±0.00	65.94±1.48	66.49±1.48	64.86±1.91
DBLP	56.27±0.62	73.02±2.22	73.87±1.35	70.62±0.97	70.90±0.88	70.13±0.77	71.73±0.94
Deezer	54.24±2.15	53.69±2.29	55.99±3.78	52.09±2.78	52.11±2.79	52.13±3.02	51.98±2.79
LastFMAsia	57.41±0.93	81.91±1.12	84.66±0.61	80.17±0.44	80.60±0.58	79.47±1.23	80.83±0.41
Minesweeper	80.00±0.00	81.12±0.37	80.08±0.04	80.27±0.16	80.30±0.13	80.13±0.09	80.46±0.15
Pubmed	69.50±1.79	76.60±0.32	77.30±0.60	76.54±0.34	76.36±0.17	77.46±0.30	76.60±0.31
Questions	97.33±0.06	97.15±0.04	97.11±0.02	97.10±0.01	97.09±0.02	97.11±0.00	97.06±0.03
Reddit	72.16±0.15	94.89±0.02	92.76±0.46	90.67±0.13	90.58±0.12	91.00±0.24	90.46±0.03
Roman	65.80±0.35	45.08±0.43	43.93±0.45	64.66±0.84	64.25±1.09	64.06±0.78	64.25±0.64
Squirrel	30.36±0.78	47.07±0.71	46.69±1.44	49.45±0.67	49.70±0.95	42.34±3.46	48.49±0.98
Texas	48.65±4.01	51.35±2.71	54.05±2.41	73.52±2.96	72.97±2.71	73.51±1.21	71.89±1.48
Tolokers	78.16±0.02	79.93±0.10	78.50±0.55	78.18±0.03	78.18±0.04	78.24±0.03	78.20±0.02
Wiki	63.79±1.77	70.09±1.51	59.63±4.16	63.08±3.61	62.96±3.68	61.10±4.36	60.56±3.62
Wisconsin	66.67±3.51	37.25±1.64	52.94±3.10	65.89±2.23	65.10±3.22	71.77±5.98	61.18±5.08
WikiCS	72.72±0.43	79.12±0.45	79.27±0.20	75.01±0.54	74.95±0.61	73.77±0.83	74.39±0.71
OGBN-Arxiv	55.50±0.23	71.74±0.29	73.65±0.11	58.58±0.11	58.68±0.17	57.79±0.56	58.62±0.05
OGBN-Products	61.06±0.08	75.79±0.12	79.45±0.59	61.19±0.23	61.31±0.20	60.28±0.80	61.60±0.10
FullCora	33.54±0.64	61.06±0.24	58.95±0.55	57.13±0.37	57.25±0.43	56.29±0.17	56.73±0.41

E Additional Experimental Results

E.1 Ablation Study on More Graph Operators

In this section, we further validate the effectiveness of GraphAny by gradually adding diverse types of graph convolution for propagating features. Specifically, we consider three types of graph convolutions:

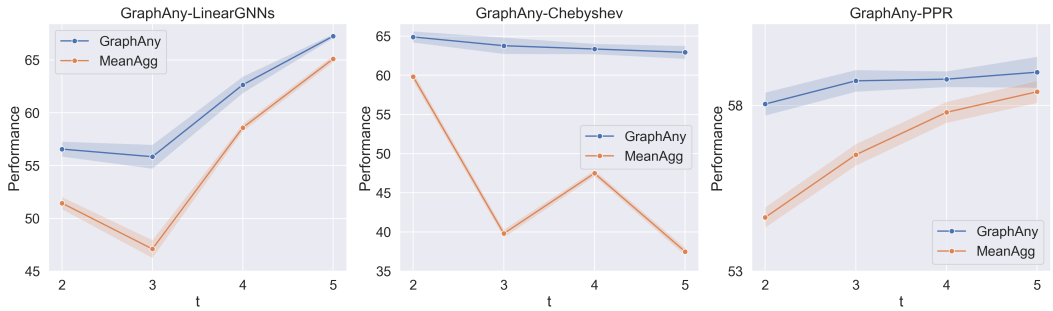


Figure 10: Ablation study with different graph operators (test accuracy in %): GraphAny-LinearGNN (left), GraphAny-Chebyshev (middle), and GraphAny-PPR (right). GraphAny is trained on the Wisconsin dataset and evaluated on all 31 datasets.

- ***GraphAny-LinearGNN*** gradually adds the following graph convolution operators: $\boldsymbol{F} = \boldsymbol{X}$ (Linear), $\boldsymbol{F} = (\boldsymbol{I} - \bar{\boldsymbol{A}})\boldsymbol{X}$ (LinearHGC1), $\boldsymbol{F} = (\boldsymbol{I} - \bar{\boldsymbol{A}})^2\boldsymbol{X}$ (LinearHGC2), $\boldsymbol{F} = \bar{\boldsymbol{A}}\boldsymbol{X}$ (LinearSGC1), and $\boldsymbol{F} = \bar{\boldsymbol{A}}^2\boldsymbol{X}$ (LinearSGC2), with $\bar{\boldsymbol{A}}$ denoting the row normalized adjaceny matrix. In this case, when increasing t high-pass and low-pass predictions are gradually added.
- ***GraphAny-Chebyshev*** gradually adds the Chebyshev filters (Defferrard et al., 2016), where $\boldsymbol{F}^{(t)}$ is computed recursively by:

$$\begin{aligned} \boldsymbol{F}^{(1)} &= \boldsymbol{X}, \\ \boldsymbol{F}^{(2)} &= \hat{\boldsymbol{L}}\boldsymbol{X}, \\ \boldsymbol{F}^{(t)} &= 2\hat{\boldsymbol{L}}\boldsymbol{F}^{(t-1)} - \boldsymbol{F}^{(t-2)}, \end{aligned} \tag{16}$$

  and $\hat{\boldsymbol{L}}$ denotes the scaled and normalized Laplacian $\frac{2\boldsymbol{L}}{\lambda_{\max}} - \boldsymbol{I}$ with $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$. In this case, when increasing t high order predictions are gradually added.
- ***GraphAny-PPR*** gradually adds the personalized PageRank with increasing restart ratios: $\{0.01, 0.25, 0.5, 0.75, 0.99\}$. In this case, when increasing t local predictions are gradually added.

We compare the performance of GraphAny against the baseline, which averages all channels, denoted as *MeanAgg*. The results are shown in Figure 10, where we have the following observations:

GraphAny consistently outperforms the baseline MeanAgg by a significant margin, demonstrating effective and consistent inductive knowledge transfer across multiple datasets with different types of graph convolutions. **The performance varies across graph convolution types.** For LinearGNNs, low-pass graph convolutions (LinearSGC1 and LinearSGC2) significantly enhance performance. In contrast, for Chebyshev graph convolutions, adding high-order convolutions reduces performance as higher-order information introduces more noise. For personalized PageRank, adding more local channels consistently improves results.

E.2 Comparison Against ACM-GNN

To comprehensively compare GraphAny's performance, we included ACM-GNN (Luan et al., 2022) as a baseline due to its good performance on both homophilic and heterophilic graphs. However, ACM-GNN faces scalability issues that prevent its application to large graphs. Using the authors' implementation[2], we encountered out-of-memory for a GPU of 40GB on four large datasets: Questions, Reddit, Arxiv, and Product. Therefore, we evaluated ACM-GNN on the remaining 27 graphs, with results reported in Table 6.

Our observations are as follows: Tuning ACM-GNN is highly time-consuming, requiring 672 GPU hours on 27 graphs, while GraphAny requires only 4 GPU hours, showcasing its efficiency and the advantage of inductive inference. In terms of performance, GraphAny outperforms ACM-SGC

[2]https://github.com/SitaoLuan/ACM-GNN/tree/main/ACM-Pytorch/

and is only slightly (1-2%) below ACM-GCN. However, this slight difference is not a significant disadvantage for GraphAny, given the unfair advantage transductive models have by leveraging the inductive validation set for parameter and hyperparameter tuning, as well as the substantial difference in runtime.

Table 6: Experimental results of ACM-GNNs. Hyperparameter search durations: ACM-GNNs (672 GPU hours), MLP/GCN/GAT (240 GPU hours total), GraphAny (denoted as Ours, 4 GPU hours).

Metric	**MLP**	**GCN**	**GAT**	**ACM-SGC**	**ACM-GCN**	**Ours-Cora**	**Ours-Wis**	**Ours-Arxiv**	**Ours-Product**
Heldout (25 graphs)	54.87	64.19	64.01	60.91	66.92	65.11	65.23	65.56	65.56
Overall (27 graphs)	55.07	63.83	64.26	61.82	67.79	65.58	66.02	66.12	66.14