GLINKX: A Scalable Unified Framework for Homophilous and Heterophilous Graphs

Marios Papachristou*
Cornell University
papachristoumarios@gmail.com

Rishab Goel
Twitter Inc.
rgoel@twitter.com

Frank Portman
Twitter Inc.
fportman@twitter.com

Matthew Miller
Twitter Inc.
mmiller@twitter.com

Rong Jin
Twitter Inc.
rj@twitter.com

Abstract

In graph learning, there have been two main inductive biases regarding graph-inspired architectures: On the one hand, higher-order interactions and message passing work well on homophilous graphs and are leveraged by GCNs and GATs. Such architectures, however, cannot easily scale to large real-world graphs. On the other hand, shallow (or node-level) models using ego features and adjacency embeddings work well in heterophilous graphs. In this work, we propose a novel scalable shallow method – GLINKX – that can work both on homophilous and heterophilous graphs. To achieve scale in large graphs, GLINKX leverages (i) novel monophilous label propagations (ii) ego/node features, (iii) knowledge graph embeddings as positional embeddings, (iv) node-level training, and (v) low-dimensional message passing. We show the effectiveness of GLINKX on several homophilous and heterophilous datasets.

1 Introduction

In recent years, graph learning methods have emerged with a strong performance for various ML tasks. Graph ML methods leverage the topology of graphs underlying the data [4] to improve their performance. Two very important design options for proposing graph ML based architectures in the context of *node classification* are related to whether the data is *homophilous* or *heterophilous*.

For homophilous data – where neighboring nodes share similar labels [26, 3] – Graph Neural Network (GNN)-based methods are able to achieve high accuracy. Specifically, a broad subclass sucessfull GNNs are Graph Convolutional Networks (GCNs) (e.g., GCN, GAT, etc.) [19, 37, 46]. In the GCN paradigm, *message passing* and *higher-order interactions* help node classification tasks in the homophilous setting since such inductive biases tend to bring the learned representations of linked nodes close to each other. However, GCN-based architectures suffer from *scalability issues*. Performing (higher-order) propagations during the training stage are hard to scale in large graphs because the number of nodes grows exponentially with the increase of the filter receptive field. Thus, for practical purposes, GCN-based methods require *node sampling*, substantially increasing their training time. For this reason, architectures [16, 41, 36, 25, 30] that leverage propagations outside of the training loop (as a preprocessing step) have shown promising results in terms of scaling to large graphs.

In *heterophilous* datasets [29], the nodes that are connected tend to have different labels. Many works that address heterophily so far can be classified into two main categories concerning scale. On

*Work done while interning at Twitter.

New Frontiers in Graph Learning Workshop at NeurIPS 2022

the one hand, recent successful architectures (in terms of accuracy) [17, 10, 43, 23, 8] that address heterophily resemble GCNs in terms of design and thus suffer from the same scalability issues. On the other hand, *shallow or node-level models* (see, e.g., [22, 44]), i.e., models that are treating graph data as tabular data and do not involve propagations during training, have shown a lot of promise for large heterophilous graphs. In [22], it is shown that combining ego embeddings (aka. node features) and adjacency embeddings works in the heterophilous setting. However, their design is still impractical in real-world data since the method (LINKX) is *not* inductive, and embedding the adjacency matrix directly requires many parameters in a model. In LINKX, the adjacency embedding of a node can alternatively be thought of as a *positional embedding* (PE) of the node in the graph, and recent developments [18, 12, 22] have elevated the importance of PEs in both homophilous and heterophilous settings. However, most of these works suggest PE parametrizations that are difficult to compute in large-scale settings. For this reason, more scalable ways of computing PEs via *knowledge graph embeddings* [13, 20, 6, 39] are useful in practical settings.

Goal & Contribution

In this work, we develop a scalable method for node classification that: (i) works both on homophilous and heterophilous graphs (ii) is *simpler and faster* than conventional message passing networks (by avoiding the neighbor sampling and message passing overhead), and (iii) can work in both a *transductive* and an *inductive*[2] setting. For a method to be scalable, we argue that it should: (i) run models on node-scale (thus leveraging i.i.d. minibatching), (ii) avoid doing message passing during training and do it a constant number of times before training, and (iii) transmit small messages along the edges. Our proposed method – GLINKX (see Section 3) – combines all the above desiderata. GLINKX has three components: (i) ego embeddings [3], (ii) PEs inspired by architectures suited for heterophilous settings, and (iii) *scalable 2nd*-hop-neighborhood propagations inspired by architectures suited for monophilous settings. Finally, we evaluate GLINKX's empirical effectiveness on several homophilous and heterophilous datasets (Section 4).

2 Preliminaries

2.1 Notation

We represent scalars with lower-case, vectors with bold lower-case letters, and matrices with bold upper-case letters. We consider a directed graph $G = G(V, E)$ with vertex set V with $|V| = n$ nodes, and edge set E with $|E| = m$ edges. Let $\boldsymbol{X} \in \mathbb{R}^{n \times d_X}$ represent the d_X dimensional node feature matrix and $\boldsymbol{P} \in \mathbb{R}^{n \times d_P}$ represent the d_P dimensional node positional embedding matrix. A node i has a feature vector $\boldsymbol{x}_i \in \mathbb{R}^{d_X}$ and a positional embedding $\boldsymbol{p}_i \in \mathbb{R}^{d_P}$ and belongs to a class $y_i \in \{1, \ldots, c\}$. The training set is denoted by V_{train}, validation set by V_{valid}, and test set by V_{test}. $\mathbb{I}\{\cdot\}$ denotes the indicator function.

2.2 Homophily, Heterophily & Monophily

Homophily and Heterophily: There are various measures of homophily in the GNN literature like node homophily and edge homophily [22]. Intuitively, homophily in a graph implies that nodes with similar labels are connected. GNN-based approaches like GCN, GAT, etc., leverage this property to improve the node classification performance. Alternatively, if a graph has low homophily – namely, nodes that connect tend to have different labels – it is said to be *heterophilous*. In other words, a graph is heterophilous if neighboring nodes do not share similar labels.

Monophily: Generally, we define a graph to be monophilous if the label of a node is similar to that of its neighbors' neighbors[4]. Etymologically, the word "monophily" is derived from the Greek words *"monos"* (unique) and *"philos"* (friend), which in our context means that a node – regardless of its label – has neighbors of primarily one label. In the context of a directed graph, monophily can be

[2]For this paper, we operate in the *transductive setting*. See App. A.1 for the inductive setting.

[3]We use ego embeddings and node features interchangeably.

[4]A similar definition of monophily has appeared in [3], whereby many nodes have extreme preferences for connecting to a certain class.

Algorithm 1 GLINKX Algorithm

Input: Graph $G(V, E)$ with train set $V_{\text{train}} \subseteq V$, node features $\boldsymbol{X}$, labels $\boldsymbol{Y}$
Output: Label Predictions $\boldsymbol{Y}_{\text{final}}$
1st Stage (KGEs). Pre-train knowledge graph embeddings $\boldsymbol{P}$ with Pytorch Biggraph.
2nd Stage (MLaP). Propagate labels and predict neighbor distribution

1. **MLaP Forward:** Calculate $\hat{\boldsymbol{y}}_i = \frac{\sum_{j \in V_{\text{train}}: j \to i} \boldsymbol{y}_j}{|\{j \in V_{\text{train}}: j \to i\}|}$ for all $i \in V_{\text{train}}$
2. **Learn distribution of a node's neighbors:**
   (a) For each epoch, calculate $\tilde{\boldsymbol{y}}_i = f_1(\boldsymbol{x}_i, \boldsymbol{p}_i; \boldsymbol{\theta}_1)$ for $i \in V_{\text{train}}$
   (b) Update the parameters s.t. $\mathcal{L}_{\text{CE},1}(\boldsymbol{\theta}_1) = \sum_{i \in V_{\text{train}}} \text{CE}(\hat{\boldsymbol{y}}_i, \tilde{\boldsymbol{y}}_i; \boldsymbol{\theta}_1)$ is maximized.
   (c) Let $\boldsymbol{\theta}_1^*$ be the parameters at the end of the training that correspond to the epoch with the best validation accuracy.
3. **MLaP Backward:** Calculate $\boldsymbol{y}'_i = \frac{\sum_{j \in V: j \to i} \tilde{\boldsymbol{y}}_j}{|\{j \in V: i \to j\}|}$ for all $i \in V_{\text{train}}$, where $\tilde{\boldsymbol{y}}_j = f_1(\boldsymbol{x}_j, \boldsymbol{p}_j; \boldsymbol{\theta}_1^*)$.

3rd Stage (Final Model). Learn a node's own distribution:

1. For each epoch, calculate $y_{\text{final},i} = f_2(\boldsymbol{x}_i, \boldsymbol{p}_i, \boldsymbol{y}'_i; \boldsymbol{\theta}_2)$.
2. Update the parameters s.t. $\mathcal{L}_{\text{CE},2}(\boldsymbol{\theta}_2) = \sum_{i \in V_{\text{train}}} \text{CE}(y_i, y_{\text{final},i}; \boldsymbol{\theta}_2)$ is maximized.

Return $\boldsymbol{Y}_{\text{final}}$

thought of as a structure that resembles Fig. 1(b) where similar nodes (in this case, three green nodes connected to a yellow node) are connected to a node with a different label.

We argue that encoding monophily into a model **can be helpful for both heterophilous and homophilous graphs (see Figs. 3(b) and 3(c)), which is one of the main motivators behind our work**. In homophilous graphs, monophily will fundamentally encode the 2nd-hop neighbor's label information, and since in such graphs, neighboring nodes have similar labels, it can provide a helpful signal for node classification. In heterophily, neighboring nodes have different labels, but the 2nd-hop neighbors may share the same label, providing helpful information for node classification. Monophily is effective for heterophilous graphs [22]. Therefore, an approach encoding monophily has an advantage over methods designed specifically for homophilous and heterophilous graphs, especially when varying levels of homophily can exist between different sub-regions in the same graph (see Section 3.3).

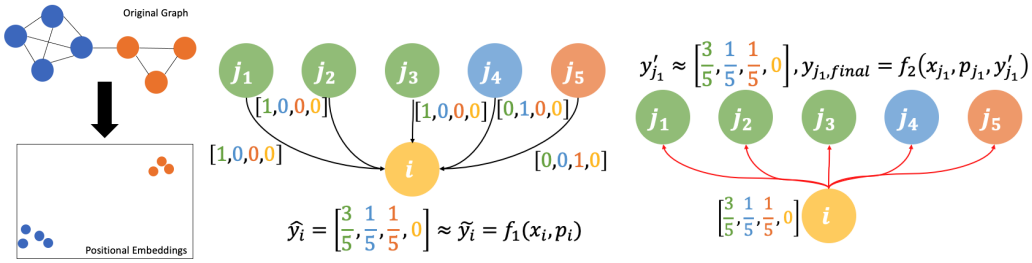
3 Our Method: GLINKX

3.1 Components & Motivation

The desiderata we laid down on Section 1 can be realized by three components: (i) PEs, (ii) ego embeddings, and (iii) label propagations that encode monophily. More specifically, ego embeddings and PEs are used as primary features, which have been shown to work for both homophilous and heterophilous graphs for the models we end up training. Finally, the propagation step is used to encode monophily to provide additional information to our final prediction.

GLINKX is described in Alg. 1 and consists of three main components: (i) PEs, (ii) ego embeddings, and (iii) propagations. Fig. 1 shows Alg. 1 on a toy graph.

Positional Embeddings: We use PEs to provide our model information about the position of each node and hypothesize that PEs are an important piece of information in the context of large-scale node classification. PEs have been used to help discriminate isomorphic graph (sub)-structures [18, 12, 35]. This is useful for both homophily [18, 12] and heterophily [22] because isomorphic (sub)-structures can exist in both the settings. In the homophilous case, adding positional information can help distinguish nodes that have the same neighborhood, but distinct position [12, 27, 38], circumventing the to do higher-order propagations [12, 21, 7] which are prone to over-squashing [2]. In heterophily, structural similarity among nodes is important for classification, as in the case of LINKX – where adjacency embedding can be considered a PE. However, in large graphs, using adjacency embeddings or Laplacian eigenvectors (as methods such as [18] suggest) can be a computational bottleneck and may be infeasible.



(a) KGEs Training (b) MLaP Forward & Neighbor Model (c) MLaP Backward & Final Model

Figure 1: Example. For node i we want to learn a model that takes i's features $\boldsymbol{x}_i \in \mathbb{R}^{d_X}$, and PEs $\boldsymbol{p}_i \in \mathbb{R}^{d_P}$ and predict a value $\tilde{\boldsymbol{y}}_i \in \mathbb{R}^c$ that matches the label distribution of it's neighbors neighbors $\hat{\boldsymbol{y}}_i$ using a shallow model. Next, we want to propagate (outside the training loop) the (predicted) distribution of a node back to its neighbors and use it together with the ego features and the PEs to make a prediction about a node's own label. We propagate $\tilde{\boldsymbol{y}}_i$ to its neighbors j_1 to j_5. For example, for j_1, we encode the propagated distribution estimate $\tilde{\boldsymbol{y}}_i$ from i to form $\boldsymbol{y}'_{j_1}$. We predict the label by using $\boldsymbol{y}'_{j_1}, \boldsymbol{x}_{j_1}, \boldsymbol{p}_{j_1}$.

In this work, we leverage *knowledge graph embeddings* (KGEs) to encode positional information about the nodes. Using KGEs has two benefits: Firstly, KGEs can be trained quickly for large graphs. This is because KGEs compress the adjacency matrix into a fixed-sized embedding, and adjacency matrices have been shown to be effective in heterophilous cases. Further, KGEs are low-dimensional compared to the adjacency matrix (e.g., $d_P \sim 10^2$), which allows for faster training and inference times. Secondly, KGEs can be pre-trained efficiently on such graphs [20] and can be used off-the-shelf for other downstream tasks, including node classification (see, e.g., [13]). So, in the 1st Stage of our methods in Alg. 1, we train the KGEs model on the available graph structure.

Here, we fix this positional encoding once they are pre-trained for downstream usage. One can fine-tune these along with learning [12] in the downstream task, but we leave this for future work. Finally, we note that this step is transductive but we can easily make it inductive (see e.g. [13, 1]).

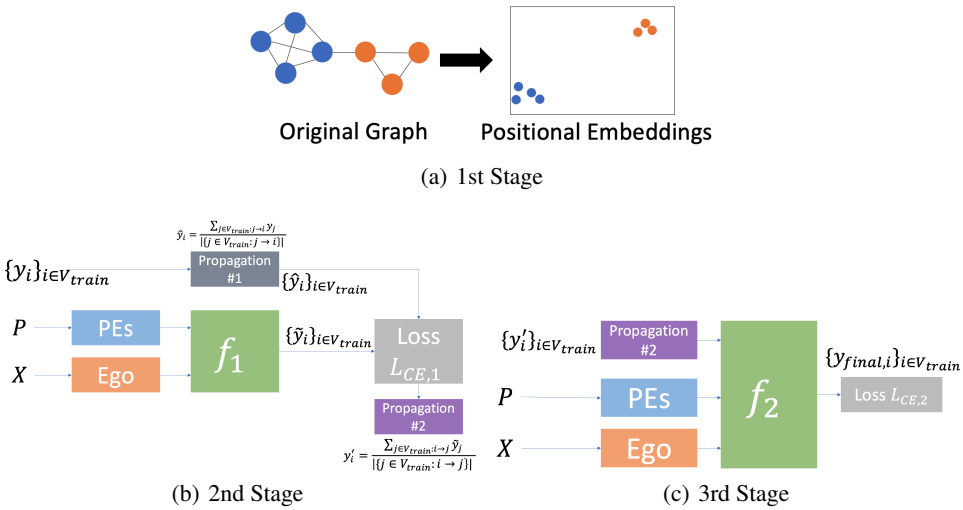
Ego Embeddings: We obtain ego embeddings from the node features. Such embeddings have been used in homophilous and heterophilous settings (e.g., [22, 46]). Node embeddings are useful for tasks where the graph structure provides little/no information about the task.

Monophilous Label Propagations: We now propose a novel monophily-inspired label propagation (see Section 2.2) which we refer to as Monophilous Label Propagation (MLaP). MLaP has the advantage that we can use it both for homophilous and heterophilous graphs or in a scenario with varying levels of graph homophily (see Section 3.3) as it encodes monophily (Section 2.2).

To understand how MLaP encodes monophily, we consider the example in Fig. 1. In this example, we have three green nodes connected to a yellow node and two nodes of different colors connected to the yellow node. Then, one way to encode monophily in Fig. 1(b) while predicting label for $j_\ell, \ell \in [5]$, is to get a *distribution* of labels of nodes connected to node i thus encoding its neighbors' distribution. The fact that there are more nodes with green color than other colors can be used by the model to make a prediction. But this information may only sometimes be present, or there may be few labeled nodes around node i. Consequently, we propose to use a model that predicts the label distribution of nodes connected to i. We use the node features ($\boldsymbol{x}_i$) and PE ($\boldsymbol{p}_i$) of node i to build this model since nodes that are connected to node i share similar labels, and thus, the features of node i must be predictive of its neighbors. So, in Fig. 1(b), we train a model to predict a distribution of i's neighbors. Next, we provide j_ℓ the learned distribution of i's neighbors by propagating the learned distribution from i back to j_ℓ. Eqs. (1) to (3) correspond to MLaP. We train a final model that leverages this information together with node features and PEs (Fig. 1(c)).

3.2 GLINKX

We put the components discussed in Section 3.1 together into three stages. In the first stage, we pre-train the PEs by using KGEs. Next, encode monophily into our model by training a model that



(a) 1st Stage

(b) 2nd Stage

(c) 3rd Stage

Figure 2: Block Diagrams of GLINKX stages.

predicts a node's neighbors' distribution and by propagating the soft labels from the fitted model. Finally, we combine the propagated information, node features, and PEs to train a final model. GLINKX is described in Alg. 1 and consists of three main components detailed as block diagrams in Fig. 2. Fig. 1 shows the GLINKX stages from Alg. 1 on a toy graph:

1st Stage (KGEs): We train KGEs as PEs by using Pytorch-Biggraph and the DistMult method [39].

2nd Stage (MLaP): First (2nd Stage in Alg. 1), for a node we want to learn the distribution of *its neighbors* and then propagate this information back to its neighbors. To achieve this, we propagate the labels from a node's neighbors, i.e. calculate

$$\hat{\boldsymbol{y}}_i = \frac{\sum_{j \in V_{\text{train}}: j \to i} \boldsymbol{y}_j}{|\{j \in V_{\text{train}} : j \to i\}|} \qquad \forall i \in V_{\text{train}}. \tag{1}$$

We train a model that predicts the distribution of neighbors, which we denote with $\tilde{\boldsymbol{y}}_i$ using the ego features $\{\boldsymbol{x}_i\}_{i \in V_{\text{train}}}$ and the PEs $\{\boldsymbol{p}_i\}_{i \in V_{\text{train}}}$ and maximize the cross-entropy with treating $\{\hat{\boldsymbol{y}}_i\}_{i \in V_{\text{train}}}$ as ground truth labels, namely we maximize

$$\mathcal{L}_{\text{CE, 1}}(\boldsymbol{\theta}_1) = \sum_{i \in V_{\text{train}}} \sum_{l \in [c]} \hat{\boldsymbol{y}}_{i,l} \log(\tilde{\boldsymbol{y}}_{i,l}), \tag{2}$$

where $\tilde{\boldsymbol{y}}_i = f_1(\boldsymbol{x}_i, \boldsymbol{p}_i; \boldsymbol{\theta}_1)$ and $\boldsymbol{\theta}_1 \in \Theta_1$ is a learnable parameter vector. Although in this paper we assume to be in the *transductive setting*, this step allows us to be inductive (see App. A.1). In **??** we give a theoretical justification of this step, namely *"why is it good to use a parametric model to predict the distribution of neighbors?"*.

Finally, we propagate the predicted soft-labels $\tilde{\boldsymbol{y}}_i$ back to the original nodes, i.e. calculate

$$\boldsymbol{y}'_i = \frac{\sum_{j \in V: j \to i} \tilde{\boldsymbol{y}}_j}{|\{j \in V : i \to j\}|} \quad \forall i \in V_{\text{train}}, \tag{3}$$

where the soft labels $\{\tilde{\boldsymbol{y}}_i\}_{i \in V_{\text{train}}}$ have been computed with the parameter $\boldsymbol{\theta}_1^*$ of the epoch with the best validation accuracy from the model $f_1(\cdot|\boldsymbol{\theta}_1)$.

3rd Stage (Final Model): As a final stage (3rd Stage in Alg. 1), we make the final predictions $\boldsymbol{y}_{\text{final, i}} = f_2(\boldsymbol{x}_i, \boldsymbol{p}_i, \boldsymbol{y}'_i; \boldsymbol{\theta}_2)$ by combining the ego embeddings, PEs, and the (back)-propagated soft labels ($\boldsymbol{\theta}_2$ is a learnable parameter vector). We use the soft-labels $\tilde{\boldsymbol{y}}_i$ instead of the actual labels

one-hot (y_i) in order to avoid label leakage, which hurts performance (see also [33] for a different way to combat label leakage). Finally, we maximize the cross-entropy with respect to a node's own labels,

$$\mathcal{L}_{\text{CE, 2}}(\boldsymbol{\theta}_2) = \sum_{i \in V_{\text{train}}} \sum_{l \in [c]} \mathbb{I}\{y_i = l\} \log(\boldsymbol{y}_{\text{final, i},l}), \tag{4}$$

Overall, Stage 2 corresponds to learning the neighbor distributions and propagating this information back to them and Stage 3 uses these distributions to train a new model which predicts a node's own labels.

Label Sparsity: In the case that the graph is very sparsely labeled, we expect that the performance from the MLaP step to go down, however, other components (PEs, node features) can be more predictive of the node's label. Also, we note that other standard methods, such as GCNs, would have decreased performance due to label sparsity.

3.3 Varying Homophily

Graphs with monophily experience homophily, heterophily, or both. For instance, in the yelp-chi dataset – where we classify a review as spam/non-spam (see Fig. 3) – we observe a case of monophily together with varying homophily. Specifically in this dataset, spam reviews are linked to non-spam reviews, and non-spam reviews usually connect to other non-spam reviews, which makes the node homophily distribution bimodal. Here the 2nd-order similarity makes both the MLaP mechanism particularly effective and PEs since we can use the PEs to distinguish nodes that have similar features but are located in different regions of the graph (homophilous/heterophilous area).

3.4 Scalability

GLINKX is highly scalable as it performs message passing a constant number of times by paying an $O(mc)$ cost, where the dimensionality of classes c is usually small (compared to d_X that GCNs rely on). In both Stages 2 and 3 of Alg. 1, we train node-level MLPs, which allow us to leverage i.i.d. (row-wise) mini batching, like tabular data, and thus our complexity is similar to other shallow methods (LINKX, FSGNN) [22, 25]. This, combined with the propagations outside the training loops, circumvent the scalability issues of GCNs. For more details, refer App. A.2.

3.5 Complementarity

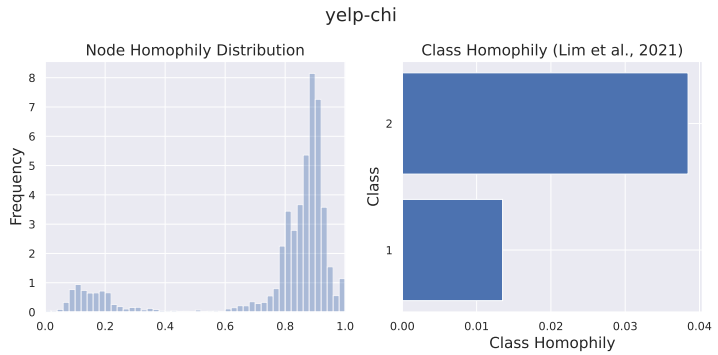
Different components of GLINKX provide a *complementary* signal to components proposed in the GNN literature [25, 41, 30]. One can combine GLINKX with existing architectures (e.g. feature propagations [25, 30], label propagations [41]) for potential metric gains. For example, SIGN computes a series of $r \in \mathbb{N}$ feature propagations $[X, \Phi X, \Phi^2 X, \ldots, \Phi^r X]$ where Φ is a matrix (e.g., normalized adjacency or normalized Laplacian) as a preprocessing step. We can include this complementary signal, namely, embed each of the propagated features and combine them in the 3rd Stage to GLINKX. Overall, although in this paper, we want to keep GLINKX simple to highlight its main components, we conjecture that adding more components to GLINKX would improve its performance on datasets with highly variable homophily.

4 Experiments

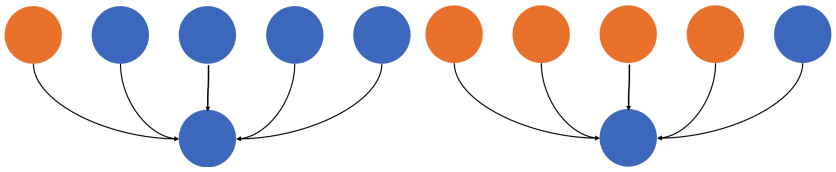
4.1 Comparisons

We experiment with homophilous and heterophilous datasets (see Tab. 1 and App. B.3). We train KGEs with Pytorch-Biggraph [20, 39]. For homophilous datasets we compare with vanilla GCN and GAT, FSGNN and Label Propagation (LP). For a fair comparison, we compare with one-layer GCN/GAT/FSGNN/LP since our method is one-hop. We also compare with higher-order (h.o.) GCN/GAT/FSGNN/LP with 2 and 3 layers. In the heterophilous case, we compare with LINKX[5]

[5] We have run our method with hyperparameter space that is a subset of the sweeps reported in [22] due to resource constraints. A bigger hyperparameter search would improve our results.



(a) Node homophily and Class homophily [22]



(b) Homophilous region example (c) Heterophilous region example

Figure 3: Top: Node and class homophily distributions for the yelp-chi dataset. Bottom: Examples of a homophilous (Fig. 3(b)) and a heterophilous (Fig. 3(c)) region in the same graph that are both monophilous, namely they are connected to many neighbors of the same kind. In a spam network, the homophilous region corresponds to many non-spam reviews connecting to non-spam reviews (which is the expected behaviour of a non-spammer user), and the heterophilous region corresponds to spam reviews targeting non-spam reviews (which is the expected behaviour of spammers), thus, yielding a graph with both homophilous and heterophilous regions such as in Fig. 3(a).

because it is scalable and is shown to work better than other baselines (e.g. H2GCN), as well as with FSGNN. Note that we do not compare GLINKX with other more complex methods because GLINKX is complementary to methods (see Section 3), and design principles from these methods can be incorporated into GLINKX. We use a *ResNet* module to combine the components from Stages 2 and 3 of our algorithm. Details about the hyperparameters we use are in App. D.

In the heterophilous datasets, GLINKX outperforms LINKX (except arxiv-year where we are within the confidence interval). Moreover, the performance gap between using KGEs and adjacency embeddings shrinks as the dataset grows. In the homophilous datasets GLINKX outperforms 1-layer GCN/GAT/LP/FSGNN and LINKX. In PubMed, GLINKX outperforms h.o. GCN/GAT and in arxiv-year GLINKX is very close to the performance of GCN/GAT.

Finally, we note that our method produces consistent results across *regime shifts*. In detail, in the heterophilous regime our method performs on par with LINKX, however when we shift to the homophilous regime, LINKX's performance drops, whereas our method's performance continues to be high. Similarly, while FSGNN performs similar to GLINKX on the homophilous datasets, we observe a big performance drop on the heterophilous datasets (see arxiv-year).

4.2 Ablation Study

We ablate each component of Alg. 1 to see each component's performance contribution. We use the hyperparameters of the best model from Tab. 1. We perform two types of ablations: (i) we remove each of the components from all stages of the training, and (ii) we remove the corresponding components only from the 3rd stage of Alg. 1. Except for removing the PEs from the 3rd stage only on ogbn-arxiv, all components contribute to increasing performance on both datasets (where adding PEs on the 1st stage improves performance).

	Homophilous Datasets		Heterophilous Datasets		
	PubMed	ogbn-arxiv	squirrel	yelp-chi	arxiv-year
n	19.7K	169.3K	5.2K	169.3K	45.9K
m	44.3K	1.16M	216.9K	7.73M	1.16M
Edge-insensitive homophily [22]	0.66	0.41	0.02	0.05	0.27
d_X/c	500 / 27	128 / 40	2089 / 5	32 / 2	128 / 5
GLINKX w/ KGEs	87.95 $\pm$0.30	69.27 $\pm$0.25	45.83 $\pm$2.89	87.82 $\pm$0.20	54.09 $\pm$0.61
GLINKX w/ Adjacency	88.03 $\pm$0.30	69.09 $\pm$0.13	69.15 $\pm$1.87	89.32 $\pm$0.45	53.07 $\pm$0.29
Label Propagation (1-hop)	83.02 $\pm$0.35	69.59 $\pm$0.00	32.22 $\pm$1.45	85.98 $\pm$0.28	43.71 $\pm$0.22
Label Propagation (2-hop)	83.44 $\pm$0.35	69.78 $\pm$0.00	43.41 $\pm$1.44	85.95 $\pm$0.26	46.30 $\pm$0.27
Label Prop. on $\mathbb{I}[A^2 - A - I \geq 0]$	82.14 $\pm$0.33	9.87 $\pm$0.00	24.43 $\pm$1.18	85.68 $\pm$0.32	23.08 $\pm$0.13
LINKX (from [22])	87.86 $\pm$0.77	67.32 $\pm$0.24	61.81 $\pm$1.80	85.86 $\pm$0.40	56.00 $\pm$1.34
LINKX (our runs)	87.55 $\pm$0.37	63.91 $\pm$0.18	61.46 $\pm$1.60	88.25 $\pm$0.24	53.78 $\pm$0.06
GCN w/ 1 Layer	86.43 $\pm$0.74	50.76 $\pm$0.20		N/A	
GAT w/ 1 Layer	86.41 $\pm$0.53	54.42 $\pm$0.10		N/A	
FSGNN w/ 1 Layer	88.93 $\pm$0.31	61.82 $\pm$0.84	64.06 $\pm$2.69	86.36 $\pm$0.36	42.86 $\pm$0.22
Higher-order GCN	86.29 $\pm$0.46	71.18 $\pm$0.27 (*)		N/A	
Higher-order GAT	86.64 $\pm$0.40	73.66 $\pm$0.11 (*)		N/A	
Higher-order FSGNN	89.37 $\pm$0.49	69.26 $\pm$0.36	68.04 $\pm$2.19	86.33 $\pm$0.30	44.89 $\pm$0.29

Table 1: Experimental results. (*) = results from the OGB leaderboard

	Ablation Type	Stages	All	Remove ego embeddings	Remove propagation	Remove PEs
Heterophilous	arxiv-year	All stages	54.09 $\pm$0.61	53.52 $\pm$0.77	50.83 $\pm$0.24	39.06 $\pm$0.35
	arxiv-year	3rd stage	54.09 $\pm$0.61	53.69 $\pm$0.65	50.83 $\pm$0.24	49.13 $\pm$1.10
Homophilous	ogbn-arxiv	All stages	69.27 $\pm$0.25	61.26 $\pm$0.33	62.70 $\pm$0.34	65.64 $\pm$0.18
	ogbn-arxiv	3rd stage	69.27 $\pm$0.25	67.60 $\pm$0.39	62.70 $\pm$0.34	69.62 $\pm$0.15

Table 2: Ablation Study. We use the hyperparameters of the best run from Tab. 1 with KGEs.

5 Conclusion & Future Work

We present GLINKX, a scalable method for node classification in homophilous and heterophilous graphs that combines three components: (i) ego embeddings, (ii) PEs, and (iii) monophilous propagations. Our method is complementary to what other methods propose since we can incorporate extra components such as feature propagations, label propagations, and attention to GLINKX. As future work, (i) GLINKX can be extended in heterogeneous graphs, (ii) use more expressive methods such as attention or Wasserstein barycenters [9] for averaging the low-dimensional messages, and (iii) add complementary signals.

Acknowledgements

The authors would like to thank Maria Gorinova, Fabrizio Frasca, Sophie Hilgard, Ben Chamberlain, and Katarzyna Janocha from Twitter Cortex Applied Research for the useful discussions and comments on our work. MP would also like to thank Felix Hohne for his help regarding questions about the LINKX codebase.

References

[1] Marjan Albooyeh, Rishab Goel, and Seyed Mehran Kazemi. Out-of-sample representation learning for knowledge graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2657–2666, 2020.

[2] Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.

[3] Kristen M Altenburger and Johan Ugander. Monophily in social networks introduces similarity among friends-of-friends. *Nature human behaviour*, 2(4):284–290, 2018.

[4] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[5] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016.

[6] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[7] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[8] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. *arXiv preprint arXiv:2006.07988*, 2020.

[9] Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 685–693, Bejing, China, 22–24 Jun 2014. PMLR.

[10] Francesco Di Giovanni, James Rowbottom, Benjamin P Chamberlain, Thomas Markovich, and Michael M Bronstein. Graph neural networks as gradient flows. *arXiv preprint arXiv:2206.10991*, 2022.

[11] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 315–324, 2020.

[12] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *arXiv preprint arXiv:2110.07875*, 2021.

[13] Ahmed El-Kishky, Thomas Markovich, Serim Park, Chetan Verma, Baekjin Kim, Ramy Eskander, Yury Malkov, Frank Portman, Sofía Samaniego, Ying Xiao, et al. Twhin: Embedding the twitter heterogeneous information network for personalized recommendation. *arXiv preprint arXiv:2202.05387*, 2022.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learningfor image recognition. *ComputerScience*, 2015.

[15] 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. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[16] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

[17] Di Jin, Rui Wang, Meng Ge, Dongxiao He, Xiang Li, Wei Lin, and Weixiong Zhang. Raw-gnn: Random walk aggregation based graph neural network. *arXiv preprint arXiv:2206.13953*, 2022.

[18] Jinwoo Kim, Tien Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *arXiv preprint arXiv:2207.02505*, 2022.

[19] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[20] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems*, 1:120–131, 2019.

[21] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019.

[22] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021.

[23] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Is heterophily a real nightmare for graph neural networks to do node classification? *arXiv preprint arXiv:2109.05641*, 2021.

[24] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

[25] Sunil Kumar Maurya, Xin Liu, and Tsuyoshi Murata. Improving graph neural networks with simple architecture design. *arXiv preprint arXiv:2105.07634*, 2021.

[26] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, pages 415–444, 2001.

[27] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

[28] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*, 2020.

[29] Everett M Rogers, Arvind Singhal, and Margaret M Quinlan. Diffusion of innovations. In *An integrated approach to communication theory and research*, pages 432–448. Routledge, 2014.

[30] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 7:15, 2020.

[31] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[32] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[33] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.

[34] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pages 243–246, 2015.

[35] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. *arXiv preprint arXiv:1910.00452*, 2019.

[36] Chuxiong Sun, Hongming Gu, and Jie Hu. Scalable and adaptive graph neural networks with self-label-enhanced training. *arXiv preprint arXiv:2104.09376*, 2021.

[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[38] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[39] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[40] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[41] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. Graph attention multi-layer perceptron. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

[42] Elena Zheleva and Lise Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *Proceedings of the 18th international conference on World wide web*, pages 531–540, 2009.

[43] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S Yu. Graph neural networks for graphs with heterophily: A survey. *arXiv preprint arXiv:2202.07082*, 2022.

[44] Zhiqiang Zhong, Sergey Ivanov, and Jun Pang. Simplifying node classification on heterophilous graphs with compatible label propagation. *arXiv preprint arXiv:2205.09389*, 2022.

[45] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. Graph neural networks with heterophily. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11168–11176, 2021.

[46] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

Supplementary Materials

A Model Addendum

A.1 Inductive Setting

In the paper we focus on the transductive setting. Here we show how we can extend our framework to the inductive setting. In the inductive setting, during training we only have access to the training graph $G_{\text{train}}(V_{\text{train}}, E_{\text{train}})$. During test, the whole graph $G \subseteq G_{\text{train}}$ is revealed and we make predictions for the test set. The stages of GLINKX in the inductive setting are as follows:

1st Stage (KGEs). For the KGE pretraining stage, we train KGEs on G_{train}. Then we can use existing methods in the literature such as [13, 1] to infer the KGEs of the test nodes.

2nd Stage (MLaP). We train the model that predicts the distribution of the neighbors on G_{train} as in Alg. 1. Then for the test nodes, we know their ego features $\boldsymbol{x}_i$ and we can also infer their PEs (see above) from the pre-trained PEs on the training set. We use these and the pre-trained shallow model in order to predict $\tilde{\boldsymbol{y}}_i$ for the test nodes. From the soft-labels $\tilde{\boldsymbol{y}}_i$ we calculate $\boldsymbol{y}'_i$

3rd Stage. First, we train the second shallow model by using the propagated soft-predictions only on G_{train}. Then, we infer the soft-labels $\tilde{\boldsymbol{y}}_i$ from the 2nd stage and propagate them back to the original nodes. We then use the ego features, PEs and the propagated information $\boldsymbol{y}'_i$ to make predictions on the test set.

Test set inference is illustrated by Alg. 2

Algorithm 2 Inductive GLINKX

Input: Graph $G(V, E)$ with train set $V_{\text{train}} \subseteq V$ and test set $V_{\text{test}} \subseteq V$, node features X, labels Y
Output: Predictions for all nodes $i \in V_{\text{test}}$

Pre-training. Call Alg. 1 with input graph $G_{\text{train}}(V_{\text{train}}, E_{\text{train}})$ and compute PEs $\{\boldsymbol{p}_i\}_{i \in V_{\text{train}}}$, and pre-trained models f_1 (from 2nd stage) and f_2 (from 3rd stage)

1st Stage (KGEs). Create PEs $\{\boldsymbol{p}_i\}_{i \in V_{\text{test}}}$ for the nodes of the test set (see e.g. [13, 1])

2nd Stage (MLaP). Predict the distribution of neighbors for the nodes of the test set as $\tilde{\boldsymbol{y}}_i = f_1(\boldsymbol{x}_i, \boldsymbol{p}_i)$ for all $i \in V_{\text{test}}$. Propagate labels backwards and train model to predict a node's own labels Calculate $\boldsymbol{y}'_i = \frac{\sum_{j \in V: j \to i} \tilde{\boldsymbol{y}}_j}{|\{j \in V: i \to j\}|}$ for all $i \in V_{\text{test}}$.

3rd Stage (Final Model). Compute $\boldsymbol{y}_{\text{final},i} = f_2(\boldsymbol{x}_i, \boldsymbol{p}_i, \boldsymbol{y}'_i)$ for all $i \in V_{\text{test}}$.

Return $\{\boldsymbol{y}_{\text{final},i}\}_{i \in V_{\text{test}}}$

A.2 Scalability

The propagations have to be done constant number (ideally once) of times by paying an $O(mc)$ cost, where the dimensionality of classes c is usually small (compared to d_X that GCNs rely on). In both stages 2 and 3 of Alg. 1 we train node-level MLPs which allow us to leverage i.i.d. (row-wise) minibatching, like tabular data, and thus our complexity is similar to other shallow methods (LINKX, FSGNN) [22, 25]; meeting our requirements. This, combined with the propagations that happen outside of the training loops, circumvent the issues of GCNs in large-scale graphs.

Cost of 1st Stage. For the first stage, we are using Pytorch-Biggraph to train PEs. Please refer to [20] for more information.

Cost of 2st Stage. For the second stage we pay **once** $O(m_{\text{train}}c)$ ($m_{\text{train}} = |E_{\text{train}}|$ is the number of edges in the graph induced by the train nodes) to run MLaP Forward and then the MLP that takes $\boldsymbol{x}_i$ and $\boldsymbol{p}_i$ with L_X layers for the features $\boldsymbol{x}_i$ and L_P layers for the PEs $\boldsymbol{p}_i$, and L_{agg} layers for the ResNet and hidden dimension h, has forward pass complexity $O\left(n(d_X h + h^2 L_X + d_P h + h^2 L_P + h^2 L_{\text{agg}})\right)$. Finally, we pay again **once** $O(m_{\text{train}}c)$ for MLaP Backward.

Cost of 3rd Stage. For the 3rd Stage, we pay $O\Big(n(d_X h + h^2 L_X + d_P h + h^2 L_P + ch + h^2 L_{\text{prop}} + h^2 L_{\text{agg}})\Big)$ for a forward pass of the MLP, where c is the class dimensionality and L_{prop} are the layers for the MLP handling the propagated labels $\boldsymbol{y}'_i$.

Comparison with LINKX. Compared to LINKX we pay an $O(m_{\text{train}}c)$ extra cost once. Regarding embedding the adjacency matrix LINKX pays a $O(mh + nh^2 L_P)$ cost for generating the h-dimensional adjacency embeddings whereas we pay $O(nd_P h + nh^2 L_P)$ which is better for $d_P = O(m/n)$. This holds in most real-world large networks since $d_P \sim 10^2$ whereas $n \sim 1\text{B}$ and $m \sim 100\text{B}$. Also note that using KGEs as PEs has an additional benefit, since KGEs can be trained only *once* and can be used off-the-shelf for other downstream tasks.

Inference Complexity (Alg. 2). On the inductive setting, given pretrained models f_1, f_2 from stages 2 and 3, the runtime Alg. 2 without the pretraining stage is $O\left(mc + n_{\text{test}}(d_X h + h^2 L_X + d_P h + h^2 L_P + h^2 L_{\text{agg}})\right)$ to generate predictions for the test nodes, where $n_{\text{test}} = |V_{\text{test}}|$.

A.3 Monophilous Label Propagation (MLaP): Motivation

In monophily, a node shares labels with its neighbors' neighbors. One way to encode monophily in Fig. 1(a) while predicting the label for j_i is to get a distribution of labels of nodes connected to node i, thus encoding its neighbors' distribution. The fact that there are more nodes with green color than other colors can be used by the model to make a prediction. But this information may only sometimes be present, or there may be few labeled nodes around node i. For this reason, we propose to use a model that predicts the label distribution of nodes connected to node i. And we use the node features ($\boldsymbol{x}_i$) and PE ($\boldsymbol{p}_i$) of node i to build this model because nodes that are connected to node i share similar labels and hence, the features of node i must be predictive of its neighbors. So, in Stage 2, for every node, we train a model to predict the distribution of its neighbors. We start with the objective to provide node j_i with the neighbors' label distribution of node i. Then, we propagate this information to j_i, thus encoding monophily. Finally, in Stage 3, we train a model that uses the node features, the PEs, and the propagated labels.

B Experiments Addendum

B.1 Implementation and Environment

GLINKX is implemented in Pytorch-Geometric. For the knowledge graph embeddings we use the official Pytorch-Biggraph implementation. We use the official implementation of LINKX. For hardware we used Vertex AI notebooks with 8 NVIDIA Tesla V100 with 16GB of memory and 96 CPUs.

B.2 Experimental Protocol

Error Bars. For the PubMed dataset and the heterophilous datasets provided by [28, 22, 40] we used the fixed splits provided with a fixed seed. For the OGB datasets, where there exists only one officialy released split for each dataset [15] we generate the error bars by runnning the respective experiments with 10 different seeds (0-9).

B.3 Datasets

For our experiments we use the following datasets (see Tab. 1):

- *PubMed [40, 32].* The PubMed dataset consists of scientific publications from PubMed database pertaining to diabetes classified into one of three classes. Each node is described by a TF-IDF weighted word vector from a dictionary which consists of 500 unique words.
- *ogbn-arxiv [15, 5, 34].* The ogbn-arxiv dataset is a directed graph, representing the citation network between all CS papers on arxiv mined from the Microsoft Academic Graph (MAG).

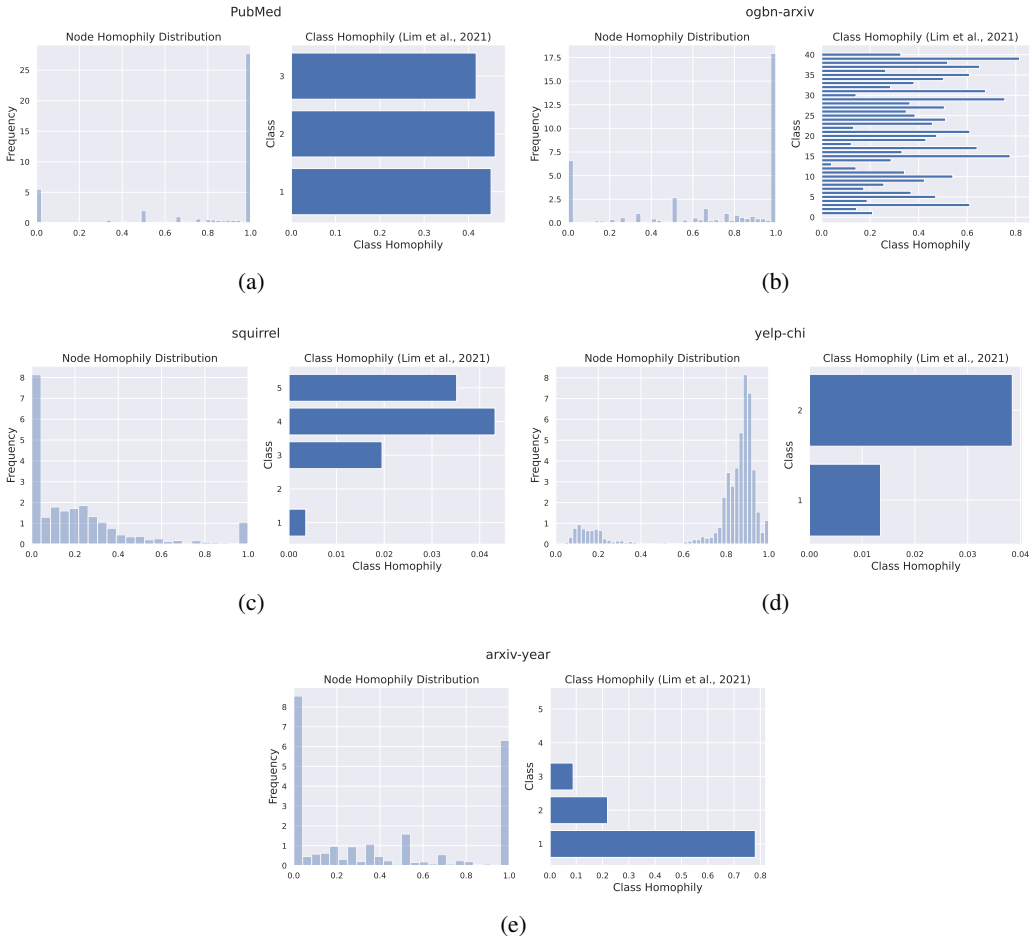


Figure 4: Node Homophily Distribution and Class (or Edge-insensitive) Homophily [22].

Nodes are papers and edges correspond to citations. The node features correspond to the average of word embeddings of the title and abstract of the papers. The task is to predict the papers' subcategories (e.g. CS.LG, CS.SI etc.).

- *squirrel [31].* The data represents page-to-page networks on squirrels mined from Wikipedia between October 2017 and November 2018. Node represent articles and edges are mutual links between the pages. The features of each node are binary and represent the existence of informative nouns that appear on the corresponding Wikipedia pages.
- *arxiv-year, yelp-chi [22].* The arXiv-year dataset is derived from the ogbn-arxiv dataset where the labels have been changed in order to convert the dataset from homophilous to heterophilous. Instead of predicting each paper's subcategories, the new task focuses on predicting the year that a paper is posted, where batches of years have been converted to labels. The yelp-chi dataset includes hotel and restaurant reviews filtered (spam) and recommended (legitimate) by Yelp. The graph structure comes from [11] and the features from [32]. The task is to predict whether a review is a spam or not.

C Further Related Work

Homophilous Methods. The backbone of graph learning is based on homophily. A graph is homophilous if connected nodes have similar labels (and usually similar features). In such settings, propagating node embeddings and then combining these embeddings with a permutation-invariant function and perhaps an edge-wise attention mechanism yields decent results [19, 37]. Recently,

much attention has been given to large-scale settings where conventional GNNs – such as GCN and GAT – cannot scale. Such methods are based on multiple propagations of the features over layers, and back-propagating through the weights is an infeasible task for large-scale networks.

For this reason, several methods have been proposed to tackle this computational bottleneck which can scale to networks with hundreds of millions of nodes. Conceptually, the most characteristic large-scale method is SIGN [30], where multiple stages of *non-trainable* filters are applied to the initial features to create node embeddings – which can be stored for later use. Then a shallow model (MLP) is applied on top of the final embeddings to produce the final predictions. Various follow-up methods tweak the SIGN architecture to add attention (SAGN [36]), softmax regularization, soft-selection, and hop-normalization (FSGNN [25]) and improve upon its performance. A related method is also GAMLP [41], where the authors combine label propagations and feature propagations and combine them with an attention mechanism. However, their method does not take into account heterophilous inductive biases.

Another important method for large-scale homophilous node classification is Correct-And-Smooth (C&S) [16]. C&S lifts the assumption that GNNs are essentially necessary for good performance on graph ML tasks. They propose to train a shallow classifier over the nodes and produce a set of *soft labels* from its predictions. Afterward, the authors leverage the graph structure and the label correlations in two steps: an error correction step that spreads residual errors in training data to correct the errors present in the test data and a prediction correlation that smooths the predictions on the test data.

Heterophilous Methods. Recently, there has been a surge in research regarding node classification in heterophilous graphs, i.e., graphs where connected nodes may have different class labels (and maybe different features). One of the first proposed methods is H2GCN [46], where the authors introduce a set of design principles for heterophilous model design. They propose the following design principles: ego and neighbor embedding separation, higher-order neighborhoods, and a combination of intermediate representations. Accordingly, they build a model – H2GCN – based on such principles and evaluate it on various real-world and synthetic datasets. Following up this work, the authors of [22] propose a significantly simpler architecture – LINKX – which combines embeddings of the ego features and embeddings of the corresponding rows of the adjacency matrix (inspired by the LINK method [42]) with a ResNet block [14]. The LINKX method is very lightweight since nodes are processed on an i.i.d. (row-wise) basis – i.e., there are no graph dependencies between the nodes – and mini-batching can be performed efficiently (contrary to propagation-based methods). LINKX is shown to outperform H2GCN on a variety of datasets. However, LINKX falls short in node classification tasks that regard homophilous datasets (such as Cora, CiteSeer, and PubMed [40]) where standard methods such as GCN and GAT yield better results.

The recent framework of [23] argues that not all cases of heterophily are harmful to GNN-based architectures and propose a set of metrics based on the construction of a similarity matrix that measures the influence of the graph and the input features on GNNs. Based on their observations, the authors design a model based on a filterbank which consists of a high-pass and a low-pass filter to address harmful heterophily achieving state-of-the-art results.

Methods for Homophily and Heterophily. Recently, there have been lots of methods that have been adapted to work both in homophily and heterophily regimes. H2GCN [46] (see above) was one of the first methods shown to work in both datasets. RAW-GNN [17] is a random-walk-based GCN that exploits both homophily and heterophily by doing random walks and aggregations in two ways: breadth-first for homophily and depth-first for heterophily. CPGNN [45] is a GCN-based architecture that uses a compatibility matrix for modeling the heterophily or homophily level in the graph, which can be learned in an end-to-end fashion, enabling it to go beyond the assumption of strong homophily. GPR-GNN [8] addresses feature over-smoothing and homophily/heterophily by combining GNNs with Generalized PageRank techniques, where each step of feature propagation is associated with a learnable weight. The amplitudes of the weights trade off the degree of smoothing of node features and the aggregation power of topological features. However, all the above methods suffer from the same scalability issues that GCNs have.

Regarding scalable methods for both homophily and heterophily, the FSGNN [25] method is also closely related to our work and relies on using propagations which are separate from the training (similarly to SIGN) and also uses separate (higher-order) feature propagations for homophily and heterophily. On the contrary, our method, in its current form, is one-hop and relies on different

components to address homophily and heterophily. Besides, their components can be added to our method, together with the ego features, the PEs, and the propagations, and thus, from another viewpoint, their work can be seen as complementary to ours. CLP [44] combines a shallow (base) predictor model with a modified Label Propagation that works both in homophily and heterophily, where they use a class compatibility matrix (similarly to CPGNN) for the Label Propagation step. Our method is fundamentally different than theirs; however, incorporating a compatibility matrix for the propagation stage constitutes interesting future work.

Positional Embeddings. Following up on the recent success of LINKX in classifying nodes in heterophilous settings based partially on the *position* of each node (adjacency embedding), a series of methods have been suggested for incorporating positional embeddings on graph methods [18, 12]. More specifically, [12] proposes the MLPGNN-LSPE architecture, which is able to simultaneously learn both structural and positional embeddings for nodes, whereas [18] proposes TokenGT, which treats all nodes and edges as independent tokens, augments them with positional embeddings – i.e., eigenvectors of the normalized Laplacian of the graph ignoring edge directions – and then feed them to a Transformer model. They prove that their framework is at least as expressive as a 2-IGN [24] composed of equivariant linear layers, which is more expressive than all message-passing GNNs. However, creating positional embeddings that require the factorization of, e.g., the Laplacian matrix to compute a set of orthonormal eigenvectors is infeasible for large datasets. For this reason, various methods have been proposed for embedding nodes in a graph in a scalable manner, such as TransE [6], DistMult [39][6]. The recent development of Pytorch-Biggraph [20] allows training such embeddings on billion-scale heterogeneous graphs such as the Twitter graph [13].

D Hyperparameters

For each of the methods we train for 200 epochs and report the test set accuracy on the epoch with the best validation accuracy.

D.1 KGEs

We use the following hyperparameters for the KGEs using Pytorch-Biggraph:

- dimension = 400
- 50 epochs
- negative samples = 1000
- batch size = 10000
- `dot` comparator, `softmax` loss [39]
- learning rate = 0.1

D.2 GLINKX w/ Adjacency

D.2.1 Sweeps

We perform the following sweeps:

- glinkx_init_layers_X $\in \{1, 2\}$
- glinkx_init_layers_A"$\in \{1, 2\}$
- glinkx_init_layers_agg $\in \{1, 2\}$
- glinkx_inner_dropout : $\in \{0.5\}$
- lr $\in \{0.1, 0.001\}$
- optimize:r: AdamW

D.3 GLINKX w/ KGEs

D.3.1 Sweeps

We perform the following sweeps for all datasets

[6]For more such methods, see [20] and the references therein.

name	init_layers_A	init_layers_X	init_layers_agg	inner_dropout	lr
arxiv-year	1	2	1	0.5	0.001
PubMed	1	2	1	0.5	0.001
squirrel	2	1	1	0.5	0.001
yelp-chi	2	2	1	0.5	0.01

Table 3: GLINKX w/ Adjacency Hyperparameters

- glinkx_init_layers_X $\in \{1, 2\}$
- glinkx_init_layers_A $\in \{1, 2\}$
- glinkx_init_layers_agg $\in \{1, 2\}$
- glinkx_inner_dropout : $\in \{0.5\}$
- lr $\in \{0.1, 0.001\}$
- optimize:r: AdamW
- biggraph_vector_length: 400

name	biggraph_vector_length	init_layers_A	init_layers_X	init_layers_agg	inner_dropout	lr
arxiv-year	400	2	1	1	0.5	0.01
ogbn-arxiv	400	2	2	2	0.5	0.001
PubMed	400	2	2	2	0.5	0.01
squirrel	400	2	1	2	0.5	0.001
yelp-chi	400	2	2	2	0.5	0.01

Table 4: GLINKX w/ Biggraph Hyperparameters

D.4 GAT

D.4.1 Sweeps

- gat_num_layers $\in \{1\}$
- gat_hidden_channels $\in \{4, 8, 16, 32\}$
- gat_num_heads $\in \{2, 4\}$
- lr $\in \{0.01, 0.001\}$

name	gat_heads	gat_hidden_channels	gat_num_layers	lr
arxiv-year	4	8	1	0.1
ogbn-arxiv	4	4	1	0.1
PubMed	4	16	1	0.1
squirrel	4	4	1	0.1

Table 5: GAT w/ 1 layer Hyperparameters

D.5 GCN

D.5.1 Sweeps

We perform the following sweeps:

- gcn_num_layers $\in \{1\}$
- gcn_hidden_channels $\in \{4, 8, 16, 32, 64\}$
- lr $\in \{0.01, 0.001\}$

name	gcn_hidden_channels	gcn_num_layers	lr
arxiv-year	64	1	0.01
ogbn-arxiv	64	1	0.01
PubMed	64	1	0.01
squirrel	64	1	0.01
yelp-chi	64	1	0.01

Table 6: GCN w/ 1 layer Hyperparameter

D.6 FSGNN

We run the following sweeps for FSGNN

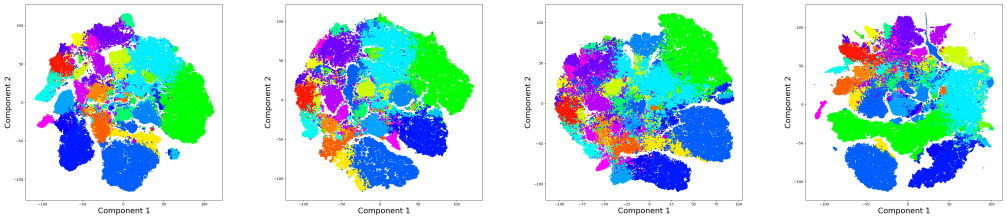
- layers = 1 for the 1-layer case and layers $\in \{2, 3\}$ for the higher-order case
- hidden channels $\in \{32, 64, 128\}$
- learning rate $\in \{0.01, 0.001\}$
- layer normalization $\in \{\text{true}, \text{false}\}$

D.7 Label Propagation

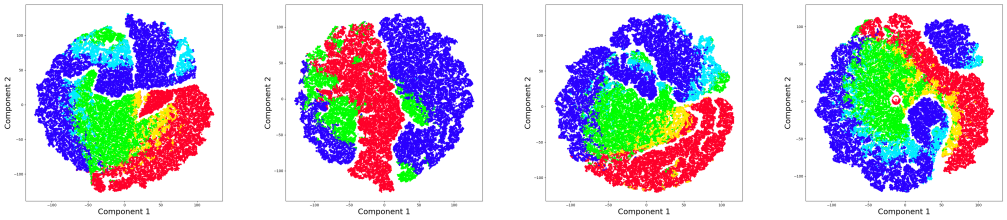
We run the following sweeps using the implementation of Label Propagation from [22]:

- $\alpha \in \{0.01, 0.1, 0.25, 0.5, 0.75, 0.99\}$
- hops $\in \{1, 2\}$

E Ablation Visualization Addendum

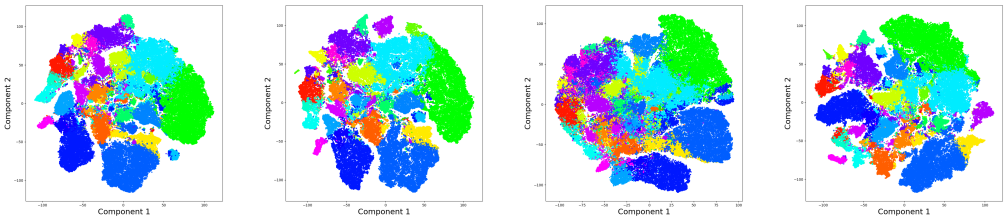


(a) All (b) Remove PEs (c) Remove propagation (d) Remove ego emb.

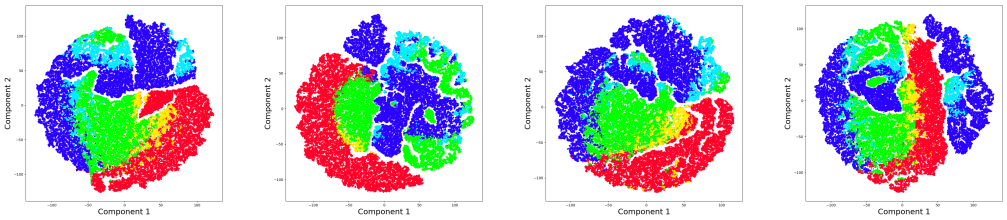


(e) All (f) Remove PEs (g) Remove propagation (h) Remove ego emb.

Figure 5: t-SNE plots of output layer embedding for ogbn-arxiv (top), arxiv-year (bottom) for ablation on both 1st and 2nd stages



(a) All (b) Remove PEs (c) Remove propagation (d) Remove ego emb.



(e) All (f) Remove PEs (g) Remove propagation (h) Remove ego emb.

Figure 6: t-SNE plots of output layer embedding for ogbn-arxiv (top), arxiv-year (bottom) for ablation on the 2nd stage