# Supplementary Material

## A Model Addendum

### A.1 Graph Mini-batching & Shallow Methods

To train a GCN-based model (or generally, whenever propagations based on the graph topology are involved in the model) on a large network (that cannot fit in the GPU memory), one has to do *minibatching* through *neighbor sampling*. For large-scale networks, mini-batching takes much longer than full-batch training, which is one of the reasons that graph GCNs are not preferred in real-world settings (see, e.g., (Jin et al., 2022b; Zhang et al., 2022a; Zheng et al., 2022a; Lim et al., 2021; Maurya et al., 2021; Rossi et al., 2020)).

For this reason, most methods that can scale on real-world settings are *shallow*. *Shallow* (or node-level) models are based on manipulating the node features $\boldsymbol{X}$ and the graph topology $\boldsymbol{A}$ so that propagations do not occur during training. Examples of these methods are LINKX, FSGNN (Maurya et al., 2021), and SIGN (Rossi et al., 2020). Such methods treat the input embeddings as tabular data and pass them through a feed-forward neural network (MLP) to produce the predictions. Thus, they avoid the need for neighborhood sampling and instead rely on simple tabular mini-batching.

### A.2 Relationship to Label Propagation

From a label propagation perspective, one could argue that our method is related to the HITS algorithm of Kleinberg (1999). Besides, on a directed graph, a potential solution would be to perform label propagation with the Gram matrix $AA^T$ and use this information as the propagated labels instead of $\boldsymbol{y}'_i$. This, however, would not be inductive.

### A.3 Scalability

Table 3: Inference Complexity (on the whole dataset) for various methods. The adjacency matrix $\boldsymbol{A}$ is given in sparse (CSR) format.

| Method | Inference Complexity |
|---|---|
| GLINKX | $O(mc + n(d_X h + h^2 L_X + d_P h + h^2 L_P + h^2 L_{\text{agg}}))$ |
| LINKX | $O(md + n(d_X h + h^2 L_X + h^2 L_{\text{agg}})$ |
| $L$-layer GCN | $O(mdL + nd^2 L)$ |
| $L$-hop SIGN/FSGNN | $O(nL(nd + d^2 h + L_{\text{agg}} h^2))$ |
| $L$-hop $\tau$-iteration Label Propagation | $O((n+m)cL\tau)$ |

***Cost of 1st Stage:*** For the 1st Stage, we are using Pytorch-Biggraph to train PEs such as TransE (Bordes et al., 2013), DistMult (Yang et al., 2014). Please refer to (Lerer et al., 2019) for more information.

***Cost of 2st Stage:*** For the 2nd 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 do perform 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 $L_{\text{agg}}$ 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, MLaP Backward costs $O(m_{\text{train}}c)$, similarly to MLaP Forward.

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

***Inference Complexity:*** Tab. 3 shows the inference complexity of GLINKX and compares it with the complexity of other methods such as LINKX, $L$-layer GCN, $L$-hop FSGNN/SIGN and LP.

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 $O(nd_Ph+nh^2L_P)$ which is better for $d_P = O(m/n)$. This holds in most real-world large networks since $d_P \sim 10^2$ whereas $n \sim 1$B and $m \sim 100$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. An $L$-layer GNN propagates using the adjacency matrix $\boldsymbol{A}$ and multiplies with a projection matrix $\boldsymbol{W}^{(i)}$ (for each layer) thus achieving an $O(dLm + nd^2L)$ complexity; the former term is due to the propagation (assuming $\boldsymbol{A}$ is sparse), and the latter term accounts for the multiplication with the projection matrix. The $L$-hop SIGN (resp. $L$-hop FSGNN) performs $L$ (resp. $2L$) propagations of the feature matrix $\boldsymbol{X}$ using a sequence of matrices $\{\Phi^\ell\}_{\ell\in[L]}$ which cost a total of $O(n^2dL)$ (assuming the matrices are dense), and then uses a ResNet with $L_{\text{agg}}$ layers block to concatenate the propagated features, which costs $O(nLdhL_{\text{agg}})$, yielding an inference complexity cost of $O(nL(nd + d^2h + L_{\text{agg}}h^2))$. Finally, Label Propagation with sparse matrices costs $O((n+m)cL\tau)$, where $\tau$ is the number of iterations of the LP algorithm.

### A.4 Knowledge Graph Embedding Training

In the sequel, we provide a general algorithm for training KGEs to be used as positional embedding $\boldsymbol{P}$. Briefly we treat $G$ as a knowledge graph with only one relation $\boldsymbol{r} = \boldsymbol{\rho}$. Specifically, each edge $(u, v) \in E$ corresponds to a head entity with embedding $\boldsymbol{h} = \boldsymbol{p}_u$ and a tail entity with embedding $\boldsymbol{t} = \boldsymbol{p}_v$. Without loss of generality we can pick $\boldsymbol{\rho} = \mathbf{1}$ to be the vector of all 1s since the constants per dimension can be absorbed by the positional embeddings.

**Algorithm 2** Knowledge Graph Embedding Training

**Input:** Graph $G(V, E)$, Embedding dimension $d_P$, Comparison function $f : \mathbb{R}^{d_P} \times \mathbb{R}^{d_P} \times \mathbb{R}^{d_P} \to \mathbb{R}$, Sample loss function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, Number of epochs $T$, Batch Size $B$, Number of negative samples $\nu$
**Output:** Positional Embeddings $\boldsymbol{P} \in \mathbb{R}^{n\times d_P}$
Randomly, initialize the entity embedding matrix $\boldsymbol{P} \in \mathbb{R}^{n\times d_P}$ with row vectors $\boldsymbol{p}_u$ for all $u \in V$.
Let $\boldsymbol{\rho} = \mathbf{1}$
For each epoch $t \in [T]$, sample minibatches $\{E_{\text{batch}}\}$ of the edges $E$ of size $B$.

1. For each minibatch $E_{\text{batch}}$ and each $(u, v) \in E$ sample $\nu$ negative samples $(u'_i, v'_i) \notin E$ and build $T_{\text{batch}} = \bigcup_{i\in[\nu]}\{((u, v), (u'_i, v'_i))\}$
2. Update the embeddings $\boldsymbol{P}$ wrt

$$\nabla_{\boldsymbol{P}} \sum_{((u,v),(u',v'))\in T_{\text{batch}}} \ell(f(\boldsymbol{p}_u, \boldsymbol{\rho}, \boldsymbol{p}_v), f(\boldsymbol{p}'_u, \boldsymbol{\rho}, \boldsymbol{p}'_v))$$

Output $\boldsymbol{P}$

In our case, we have chosen the DistMult embeddings Yang et al. (2014) which correspond to choosing the triple product

$$f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) = \sum_{i\in d_P} h_i r_i t_i$$

as a comparison function, and the margin-based ranking loss $\ell(f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}), f(\boldsymbol{h}', \boldsymbol{r}, \boldsymbol{t}')) = \max\{0, \gamma + f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) - f(\boldsymbol{h}', \boldsymbol{r}, \boldsymbol{t}')\}$ where $\gamma$ is the margin.

## B Inductive GLINKX

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.** For the KGE pretraining Stage, we train KGEs on $G_{\text{train}}$. Then we can use existing methods in the literature such as (El-Kishky et al., 2022; Albooyeh et al., 2020) to infer the KGEs of the test nodes.

**2nd Stage.** We train the model that predicts the distribution of the neighbors on $G_{\text{train}}$ 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. Then, we run MLaP backward to compute $\boldsymbol{y}'_i$.

**3rd Stage.** First, we train the second shallow model by using the propagated soft-predictions only on $G_{\text{train}}$. We then use the ego features, PEs and the propagated information $\boldsymbol{y}'_i$ to make predictions on the test set.

Alg. 3 shows the process of making predictions on the test set.

**Algorithm 3** 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.** Create PEs $\{\boldsymbol{p}_i\}_{i \in V_{\text{test}}}$ for the nodes of the test set (see e.g. (El-Kishky et al., 2022; Albooyeh et al., 2020))

**2nd Stage.** 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 $i \in V_{\text{test}}$. Calculate $\boldsymbol{y}'_i = \frac{\sum_{j \in V:(i,j) \in E} \tilde{\boldsymbol{y}}_j}{|\{j \in V:(i,j) \in E\}|}$ for all $i \in V_{\text{test}}$ (MLaP Backward).

**3rd Stage.** 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 $\{\boldsymbol{y}_{\text{final},i}\}_{i \in V_{\text{test}}}$

# C 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.

## C.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 (Yang et al., 2014)
- learning rate = 0.1

## C.2 GLINKX W/ ADJACENCY

### C.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

| 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 4: GLINKX w/ Adjacency Hyperparameters

### C.3 GLINKX w/ KGEs

#### C.3.1 SWEEPS

We perform the following sweeps for all datasets

- 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 5: GLINKX w/ Biggraph Hyperparameters

### C.4 GAT

#### C.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 6: GAT w/ 1 layer Hyperparameters

### C.5 GCN

#### C.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 7: GCN w/ 1 layer Hyperparameter

### C.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 $\in \{\text{true}, \text{false}\}$

### C.7 LABEL PROPAGATION

We run the following sweeps using the implementation of Label Propagation from (Lim et al., 2021):

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

## D EXPERIMENTS ADDENDUM

### D.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 LINK. For hardware we used Vertex AI notebooks with 8 NVIDIA Tesla V100 with 16GB of memory and 96 CPUs.

### D.2 EXPERIMENTAL PROTOCOL

**Error Bars.** For the PubMed dataset and the heterophilous datasets provided by (Pei et al., 2020; Lim et al., 2021; Yang et al., 2016) 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 (Hu et al., 2020) we generate the error bars by runnning the respective experiments with 10 different seeds (0-9).

### D.3 DATASETS

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

- *PubMed (Yang et al., 2016; Sen et al., 2008).* 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 (Hu et al., 2020; Bhatia et al., 2016; Sinha et al., 2015).* 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). 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 (Rozemberczki et al., 2021).* 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 (Lim et al., 2021).* 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 (Dou et al., 2020) and the features from (Sen et al., 2008). The task is to predict whether a review is a spam or not.

# E STAGES DIAGRAM

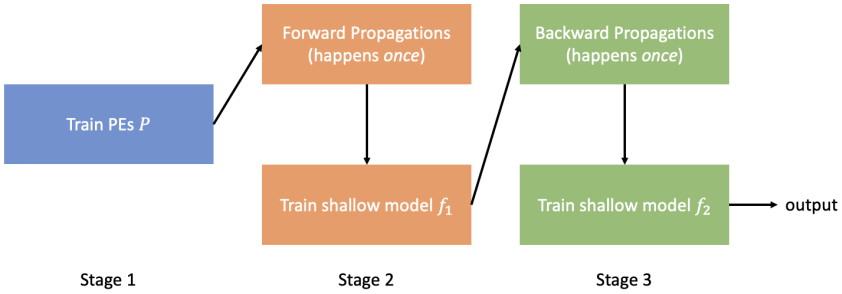


Figure 4: GLINKX stages.

# F PROOFS

## F.1 PROOF OF THM. 1 (PARAMETRIC ESTIMATION OF $Q$)

**Counting Estimator.** As the first step, we can estimate $Q_i$, denoted by $\widehat{Q}_i$ by taking the average class assignment in its neighbor $\mathcal{N}(i)$, i.e.

$$\widehat{Q}_{i,j} = \frac{1}{|\mathcal{N}(i)|} \sum_{k \in \mathcal{N}(i)} \mathbb{I}\{y_k = j\}$$

Evidently, $\mathbb{E}[\widehat{Q}_{i,j}] = Q_{i,j}$. Also, using Hoeffiding concentration inequality, we have, with a probability at least $1 - \delta$,

$$\left| \widehat{Q}_{i,j} - \frac{1}{|\mathcal{N}(i)|} \sum_{k \in \mathcal{N}(i)} P_{k,j} \right| \leq \sqrt{\frac{1}{2K} \log \frac{2}{\delta}}$$

Since

$$\boldsymbol{Q}_i = \frac{1}{|\mathcal{N}(i)|} \sum_{k \in \mathcal{N}(i)} \boldsymbol{P}_k$$

we have

$$\|\widehat{\boldsymbol{Q}}_i - \boldsymbol{Q}_i\|_\infty \leq \sqrt{\frac{1}{2K}\log\frac{2c}{\delta}}$$

with probability at least $1-\delta$. Therefore, by the law of total expectation and by minimizing wrt $\delta$, we get

$$\max_{j\in[c]} \mathbb{E}[|Q_{ij} - \widehat{Q}_{ij}|] \leq \mathbb{E}[\|\boldsymbol{Q}_i - \widehat{\boldsymbol{Q}}_i\|_\infty] \leq \min_{\delta\in(0,1)} \sqrt{\frac{1}{2K}\log\frac{2c}{\delta}} + \delta \leq O\left(\sqrt{\frac{\log(Kc)}{K}}\right)$$

**Parametric Estimator.** As the first step, we show that through a parametric optimization, we will be able to obtain a better estimation of $\boldsymbol{Q}_i$. In particular, we assume that all $\boldsymbol{Q}_i$ can be written in the parametric form $q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta})$, where $\boldsymbol{\theta}\in\mathbb{R}^D$ is the parameter of the model. Let $\boldsymbol{\theta}_*$ be the optimal model parameter such that $Q_{i,j} = q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_*)$. We define the following loss function

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} Q_{i,j}\log q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta})$$

Evidently, $\boldsymbol{\theta}_*$ is the minimizer of $\mathcal{L}(\boldsymbol{\theta})$. We assume that $\mathcal{L}(\boldsymbol{\theta})$ is $L$ smooth and $\gamma$-PL function, i.e.

$$\|\nabla\mathcal{L}(\boldsymbol{\theta})\| \leq L, \quad \mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}_*) \leq \frac{1}{2\gamma}\|\nabla\mathcal{L}(\boldsymbol{\theta})\|^2$$

The real objective function $\widehat{\mathcal{L}}(\boldsymbol{\theta})$ based on the observation of class assignments is given as

$$\widehat{\mathcal{L}}(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} \widehat{Q}_{i,j}\log q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta})$$

To find the optimal $\boldsymbol{\theta}$, we run stochastic gradient descent. At each iteration $t$, we sample one node $i_t$, and compute the gradient $\boldsymbol{g}_t$ as

$$\boldsymbol{g}_t = \sum_{j=1}^{c} \widehat{Q}_{i,j}\nabla_{\boldsymbol{\theta}}\log q(j|x_{i_t};\boldsymbol{\theta})$$

and update the solution by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\boldsymbol{g}_t$$

where $\eta$ is a fixed step size whose value will be decided later. We have

$$\begin{aligned}
&\mathbb{E}[\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_t)] \\
&\leq \mathbb{E}\left[\langle\nabla\mathcal{L}(\boldsymbol{\theta}_t), \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\rangle\right] + \frac{L}{2}\mathbb{E}\left[\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|^2\right] \\
&= -\eta\left(1 - \frac{\eta L}{2}\right)\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta}_t)\|^2\right] + \frac{\eta^2 L}{2}\mathbb{E}\left[\|\boldsymbol{g}_t - \boldsymbol{g}_t'\|^2 + \|\boldsymbol{g}_t' - \nabla\mathcal{L}(\boldsymbol{\theta}_t)\|^2\right]
\end{aligned}$$

where $\boldsymbol{g}_t' = \sum_{j=1}^{c} Q_{i,j}\nabla_{\boldsymbol{\theta}}\log q(j|x_{i_t};\boldsymbol{\theta})$. We assume that $\|\nabla_{\boldsymbol{\theta}}\log q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta})\| \leq G$ for any $\boldsymbol{\theta}$ and any $i$. We have

$$\mathbb{E}\left[\|\boldsymbol{g}_t - \boldsymbol{g}_t'\|^2\right] \leq \left(\frac{c^2}{K}\log\frac{2}{\delta} + \delta\right)G^2, \quad \mathbb{E}\left[\|\boldsymbol{g}_t' - \nabla\mathcal{L}(\boldsymbol{\theta}_t)\|^2\right] \leq G^2$$

By choosing $\delta = c^2/K$, we have

$$\mathbb{E}\left[\|\boldsymbol{g}_t - \boldsymbol{g}_t'\|^2\right] \leq \frac{2c^2G^2}{K}\log\frac{2K}{c^2}$$

Putting the above bounds together, we have

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_t)\right] \leq -\eta\left(1 - \frac{\eta L}{2}\right)\mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta}_t)\|^2\right] + \frac{\eta^2 G^2 L}{2}\left(1 + \frac{2c^2}{K}\log\frac{2K}{c^2}\right)$$

Assume $\eta L \leq 1$. Using the fact

$$\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}_*) \leq \frac{1}{2\gamma}\|\nabla \mathcal{L}(\boldsymbol{\theta}_t)\|^2$$

we have

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_*)\right] \leq \left(1 - \frac{\eta\gamma}{4}\right)\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_t) - \mathcal{L}(\boldsymbol{\theta}_*)\right] + \frac{\eta^2 G^2 L}{2}\left(1 + \frac{2c^2}{K}\log\frac{2K}{c^2}\right)$$

and therefore

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_{t+1}) - \mathcal{L}(\boldsymbol{\theta}_*)\right] \leq \exp\left(-\frac{\eta\gamma t}{4}\right)\left(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_*)\right) + \frac{\eta G^2 L}{2}\left(1 + \frac{2c^2}{K}\log\frac{2K}{c^2}\right)$$

Let $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_{n+1}$. By choosing

$$\eta = \frac{4}{n\gamma}\log\left(\frac{n\gamma(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_*))}{2G^2 L(1 + 2c^2/K\log(2K/c^2))}\right)$$

we have

$$\mathbb{E}\left[\mathcal{L}(\boldsymbol{\theta}_1) - \mathcal{L}(\boldsymbol{\theta}_*)\right] \leq \frac{2G^2 L}{n\gamma}\left(1 + \frac{2c^2}{K}\log\frac{2K}{c^2}\right)\log\frac{n\gamma(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_*))}{2G^2 L(1 + 2c^2/K\log(2K/c^2))} = O\left(\frac{\log n}{n\gamma}\right)$$

Since

$$\mathcal{L}(\boldsymbol{\theta}_1) - \mathcal{L}(\boldsymbol{\theta}_*) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_*)\log\frac{q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_1)}{q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_*)} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c}\mathrm{KL}\left(q(\cdot|\boldsymbol{\xi}_i;\boldsymbol{\theta}_*)\|q(\cdot|\boldsymbol{\xi}_i;\boldsymbol{\theta}_1)\right)$$

and (by Pinkser's Inequality),

$$\mathrm{KL}\left(q(\cdot|\boldsymbol{\xi}_i;\boldsymbol{\theta}_*)\|q(\cdot|\boldsymbol{\xi}_i;\boldsymbol{\theta}_1)\right) \geq 2\left(\sum_{j=1}^{c}|q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_1) - q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_*)|\right)^2$$

we have

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[\left(\sum_{j=1}^{c}|q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_1) - Q_{i,j}|\right)^2\right] \leq \frac{C\log n}{n\gamma}$$

where

$$C = G^2 L\left(1 + \frac{2c^2}{K}\log\frac{2K}{c^2}\right)\log\frac{\gamma(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_*))}{2G^2 L(1 + 2c^2/K\log(2K/c^2)}$$

Hence, on average, we have

$$|q(j|\boldsymbol{\xi}_i;\boldsymbol{\theta}_1) - Q_{i,j}| \leq \sqrt{\frac{C\log n}{n\gamma}}$$

which could be significantly smaller than $O(1/\sqrt{K})$ when $n \gg K$, indicating that through the parametric modeling of $Q_{i,j}$ and optimization of $\widehat{\mathcal{L}}(\boldsymbol{\theta})$, we will be able to achieve significantly better estimation of $Q_{i,j}$ than a simple counting. □

### F.2 Proof of Thm. 2 (Parametric Estimation of $P$)

**Naïve Approach.** We now will utilize the estimation $q(\cdot|\boldsymbol{\xi}_i;\boldsymbol{\theta}_1)$ further estimate $\boldsymbol{P}_i$. Similar to last section, we introduce a parametric estimator $p(j|\boldsymbol{\xi}_i;\boldsymbol{w})$, where $w \in \mathbb{R}^d$ is the parameter. We denote by $\boldsymbol{w}_*$ the optimal parameter, i.e. $P_{i,j} = p(j|\boldsymbol{\xi}_i;\boldsymbol{w}_*)$. Let $\mathcal{G}(\boldsymbol{w})$ be the objective function based on population average, i.e.

$$\mathcal{G}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} P_{i,j}\log p(j|\boldsymbol{\xi}_i;\boldsymbol{w})$$

Evidently, $w_*$ is a minimizer of $\mathcal{G}(\boldsymbol{w})$. Similar to the last Section, we assume $\mathcal{G}(\boldsymbol{w})$ is $L$ smooth and $\gamma$-PL function.

A straightforward approach for optimizing $\mathcal{G}(\boldsymbol{w})$ is, at each iteration $t$, sample a node $i_t$ from the network, and compute the stochastic gradient as

$$\boldsymbol{g}_t = y_{i_t} \nabla_{\boldsymbol{\theta}} \log p(y_{i_t} | \boldsymbol{\xi}_{i_t}; \boldsymbol{\theta}_t)$$

and update the solution as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}_t$$

Following the same analysis from the last Section, we have

$$\mathbb{E}\left[\mathcal{G}(\boldsymbol{w}_{n+1}) - \mathcal{G}(\boldsymbol{w}_*)\right] \leq O\left(\frac{\log n}{n\gamma}\right)$$

and

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}\left[\sum_{j=1}^{c} |p(j|\boldsymbol{\xi}_i; \boldsymbol{w}_{n+1}) - P_{i,j}|^2\right] \leq O\left(\sqrt{\frac{\log n}{n\gamma}}\right)$$

Here, we propose a different version of SGD, with the aim to reduce the impact of variance due to the random sample of class label $y_i$. We divide optimization into two phase. In the first phase, we will optimize $\boldsymbol{w}$ with respect to the following objective function

$$\widehat{\mathcal{G}}(\boldsymbol{w}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c} \underbrace{\left(\frac{1}{|\mathcal{N}(i)|}\sum_{k\in\mathcal{N}(i)} q(j|\boldsymbol{\xi}_k; \boldsymbol{\theta}_1)\right)}_{:=\widehat{P}_{i,j}} \log p(j|\boldsymbol{\xi}_i; \boldsymbol{w})$$

In the second phase, we will run the SGD optimization that mixes stochastic gradient with the gradient of $\widehat{\mathcal{G}}(\boldsymbol{w})$, i.e. optimizes the objective $\lambda\widehat{\mathcal{G}}(\boldsymbol{w}) + (1-\lambda)\mathcal{G}(\boldsymbol{w})$.

**Phase I.** In Phase I, we will update the solution by

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla \widehat{\mathcal{G}}(\boldsymbol{w}_t)$$

Using the standard analysis, we have

$$\begin{aligned}
&\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w}_t)] \\
&\leq -\eta\mathbb{E}\left[\langle \nabla\mathcal{G}(\boldsymbol{w}_t), \widehat{\mathcal{G}}(\boldsymbol{w}_t)\rangle\right] + \frac{\eta^2 L}{2}\|\nabla\widehat{\mathcal{G}}(\boldsymbol{w}_t)\|^2 \\
&= -\frac{\eta}{2}\mathbb{E}[|\nabla\mathcal{G}(\boldsymbol{w}_t)|^2] - \frac{\eta}{2}(1-\eta L)\mathbb{E}[\|\nabla\widehat{\mathcal{G}}(\boldsymbol{w})\|^2] + \frac{\eta}{2}\mathbb{E}[\|\nabla\mathcal{G}(\boldsymbol{w}_t) - \nabla\widehat{\mathcal{G}}(\boldsymbol{w}_t)\|^2]
\end{aligned}$$

Since

$$\begin{aligned}
&\mathbb{E}\left[\|\nabla\mathcal{G}(\boldsymbol{w}_t) - \nabla\widehat{\mathcal{G}}(\boldsymbol{w}_t)\|^2\right] | \\
&= \mathbb{E}\left\|\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{c}(P_{i,j} - \widehat{P}_{i,j})\nabla\log p(j|\boldsymbol{\xi}_i; \boldsymbol{w}_t)\right\|^2 \leq \frac{G^2}{n^2}\mathbb{E}\left|\sum_{i=1}^{n}\|\boldsymbol{P}_i - \widehat{\boldsymbol{P}}_i\|_1\right|^2 \leq \frac{CG^2\log n}{n\gamma}
\end{aligned}$$

and by further assuming $\eta L \leq 1$, we have

$$\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w}_t)] \leq -\frac{\eta}{2}\mathbb{E}[\|\nabla\mathcal{G}(\boldsymbol{w}_t)\|^2] + \frac{\eta CG^2\log n}{n\gamma}$$

or

$$\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w}_*)] \leq (1-\eta\gamma)\mathbb{E}[\mathcal{G}(\boldsymbol{w}_t) - \mathcal{G}(\boldsymbol{w}_*)] + \frac{\eta CG^2\log n}{n\gamma}$$

By taking the telescope sum, we have

$$\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1} - \mathcal{G}(\boldsymbol{w}_*)] \leq \exp\left(-\gamma\eta t\right)\left(\mathcal{G}(\boldsymbol{w}_0) - \mathcal{G}(\boldsymbol{w}_*)\right) + \frac{CG^2\log n}{n\gamma}$$

By choosing step size $\eta = 1/L$, and setting $t$ as

$$t = \frac{L}{\gamma} \log \frac{n\gamma(\mathcal{G}(\boldsymbol{w}_0) - \mathcal{G}(\boldsymbol{w}_*))}{CG^2 \log n}$$

we obtain the solution $w'$ with guarantee

$$\mathbb{E}[\mathcal{G}(\boldsymbol{w}') - \mathcal{G}(\boldsymbol{w}_*)] \leq \frac{2CG^2 \log n}{n\gamma}$$

**Phase II.** In the second phase, we use $\boldsymbol{w}'$ as the initial solution for $\boldsymbol{w}$, and ran a SGD. At each iteration $t$, we randomly sample a node $i_t$ from the graph, compute the gradient as

$$\boldsymbol{g}_t = (1-\lambda) y_{i_t} \nabla \log p(y_{i_t} | x_{i_t}; \boldsymbol{w}_t) + \lambda \nabla \widehat{\mathcal{G}}(\boldsymbol{w}_t)$$

and update the solution as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \boldsymbol{g}_t$$

Following the analysis from the last section, we have

$$\begin{aligned}
&\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w})] \\
\leq\ & -\eta \mathbb{E}\left[\langle \nabla \mathcal{G}(\boldsymbol{w}_t), (1-\lambda)\nabla \mathcal{G}(\boldsymbol{w}_t) + \lambda \widehat{\mathcal{G}}(\boldsymbol{w}_t)\rangle\right] + \frac{\eta^2 L}{2} \mathbb{E}\left[\|\lambda \nabla \widehat{\mathcal{G}}(\boldsymbol{w}_t) + (1-\lambda)\mathcal{G}(\boldsymbol{w}_t)\|^2\right] + \frac{\eta^2 \eta^2 G^2 L}{2} \\
\leq\ & -\frac{\eta}{2} \mathbb{E}\left[\|\nabla \mathcal{G}(\boldsymbol{w}_t)\|^2\right] - \frac{\eta}{2}(1-\eta L)\, \mathbb{E}\left[\|\lambda \nabla \widehat{\mathcal{G}}(\boldsymbol{w}_t) + (1-\lambda)\nabla \mathcal{G}(\boldsymbol{w}_t)\|^2\right] + \frac{\eta \lambda^2}{2} \mathbb{E}\left[\|\nabla \mathcal{G}(\boldsymbol{w}_t) - \nabla \widehat{\mathcal{G}}(\boldsymbol{w}_t)\|^2\right] \\
& + \frac{(1-\lambda)^2 \eta^2 G^2 L}{2}
\end{aligned}$$

By choosing $\eta \leq 1/L$, we have

$$\mathbb{E}\left[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w}_t)\right] \leq -\frac{\eta}{2} \mathbb{E}\left[\|\nabla \mathcal{G}(\boldsymbol{w}_t)\|^2\right] + \frac{(1-\lambda)^2 \eta^2 G^2 L}{2} + \frac{\eta \lambda^2}{2} \mathbb{E}\left[\|\mathcal{G}(\boldsymbol{w}_t) - \widehat{\mathcal{G}}(\boldsymbol{w}_t)\|^2\right]$$

$$\begin{aligned}
&\|\nabla \mathcal{G}(\boldsymbol{w}_t) - \nabla \widehat{\mathcal{G}}(\boldsymbol{w}_t)\|^2 \\
=\ & \left\| \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{c} (P_{i,j} - \widehat{P}_{i,j}) \nabla \log p(j | \boldsymbol{\xi}_i; \boldsymbol{w}_t) \right\|^2 \leq \frac{G^2}{n^2} \left| \sum_{i=1}^{n} \|\boldsymbol{P}_i - \widehat{\boldsymbol{P}}_i\|_1 \right|^2 \leq \frac{CG^2 \log n}{n\gamma}
\end{aligned}$$

we have

$$\begin{aligned}
&\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w}_t)] \\
\leq\ & -\frac{\eta}{2} \mathbb{E}[\|\nabla \mathcal{G}(\boldsymbol{w}_t)\|^2] + \frac{(1-\lambda)^2 \eta^2 G^2 L}{2} + \frac{\eta C G^2 \lambda^2 \log n}{n\gamma}
\end{aligned}$$

Using the standard analysis, we have

$$\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w}_*)] \leq \exp\left(-\eta\gamma t\right) \left(\mathcal{G}(\boldsymbol{w}') - \mathcal{G}(\boldsymbol{w}_*)\right) + \frac{CG^2 \lambda^2 \log n}{n\gamma} + \frac{(1-\lambda)^2 \eta G^2 L}{2}$$

By minimizing over $\lambda$, we have

$$\begin{aligned}
\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{t+1}) - \mathcal{G}(\boldsymbol{w}_*)] &\leq \exp\left(-\eta\gamma t\right) \left(\mathcal{G}(\boldsymbol{w}') - \mathcal{G}(\boldsymbol{w}_*)\right) + G^2 \sqrt{\frac{CL\eta \log n}{2n\gamma}} \\
&\leq \frac{2CG^2 \log n}{n\gamma} \exp(-\eta\gamma t) + G^2 \sqrt{\frac{CL\eta \log n}{2n\gamma}}
\end{aligned}$$

With $t = n$ and choosing

$$\eta = O\left(\frac{\log \log n}{n\gamma}\right)$$

we have

$$\mathbb{E}[\mathcal{G}(\boldsymbol{w}_{n+1}) - \mathcal{G}(\boldsymbol{w}_*)] \leq O\left(\frac{1}{n\gamma} \sqrt{\log n \log \log n}\right)$$

Comparing the naive approach where we simply train over class assignment $y_i$, we are able to reduce the error by a factor of $\sqrt{\log n / \log \log n}$. □

## G FURTHER RELATED WORK

**Methods for Homophily and Heterophily.** Many methods have been adapted to work both in homophilous and heterophilous regimes. H2GCN (Zhu et al., 2020) is one of the first methods shown to work in both kinds of datasets. RAW-GNN (Jin et al., 2022a) 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 (Zhu et al., 2021) 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 (Chien et al., 2020) 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 (Maurya et al., 2021) method is closely related to our work. It relies on using propagations that are separate from the training (similar to SIGN) and uses separate (higher-order) feature propagations for homophily and heterophily. In contrast, our one-hop method relies on different components to address homophily and heterophily. Furthermore, FSGNN components can be added to our method, together with the ego features, the PEs, and the propagations, and thus their work can also be seen as complementary to ours. CLP (Zhong et al., 2022) combines a shallow (base) predictor model with a modified Label Propagation that works both in homophily and heterophily by using a class compatibility matrix (similarly to CPGNN) for the Label Propagation step. Our method is fundamentally different; 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 (Kim et al., 2022; Dwivedi et al., 2021). More specifically, (Dwivedi et al., 2021) proposes the MLPGNN-LSPE architecture, which can simultaneously learn both structural and positional embeddings for nodes, whereas (Kim et al., 2022) proposes TokenGT, which treats all nodes and edges as independent tokens, augments them with positional embeddings – eigenvectors of the normalized Laplacian of the graph ignoring edge directions – and then feeds them to a Transformer model. However, creating positional embeddings that require the factorization of the Laplacian matrix is impractical for large-scale graphs. For this reason, various methods have been proposed for embedding nodes in a graph in a scalable manner, such as TransE (Bordes et al., 2013), DistMult (Yang et al., 2014)[7]. The recent development of Pytorch-Biggraph (Lerer et al., 2019) allows training KGEs large heterogeneous graphs (El-Kishky et al., 2022).

[7]For more such methods see (Lerer et al., 2019) and the references therein.