/ˈhəʊməfaɪlɪ/

Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs

Jiong Zhu
University of Michigan
jiongzhu@umich.edu

Yujun Yan
University of Michigan
yujunyan@umich.edu

Lingxiao Zhao
Carnegie Mellon University
lingxia1@andrew.cmu.edu

Mark Heimann
University of Michigan
mheimann@umich.edu

Leman Akoglu
Carnegie Mellon University
lakoglu@andrew.cmu.edu

Danai Koutra
University of Michigan
dkoutra@umich.edu

Abstract

[ˈhetərəfail]

We investigate the representation power of graph neural networks in the semi-supervised node classification task under *heterophily* or *low homophily*, i.e., in networks where connected nodes may have *different* class labels and *dissimilar* features. Many popular GNNs fail to generalize to this setting, and are even outperformed by models that ignore the graph structure (e.g., multilayer perceptrons). Motivated by this limitation, we identify a set of key designs—ego- and neighbor-embedding separation, higher-order neighborhoods, and combination of intermediate representations—that boost learning from the graph structure under heterophily. We combine them into a graph neural network, H_2GCN, which we use as the base method to empirically evaluate the effectiveness of the identified designs. Going beyond the traditional benchmarks with strong homophily, our empirical analysis shows that the identified designs increase the accuracy of GNNs by up to 40% and 27% over models without them on synthetic and real networks with heterophily, respectively, and yield competitive performance under homophily.

1 Introduction

We focus on the effectiveness of graph neural networks (GNNs) [42] in tackling the semi-supervised node classification task in challenging settings: the goal of the task is to infer the unknown labels of the nodes by using the network structure [44], given partially labeled networks with node features (or attributes). Unlike most prior work that considers networks with strong homophily, we study the representation power of GNNs in settings with *different levels of homophily* or *class label smoothness*.

Homophily is a key principle of many real-world networks, whereby linked nodes often belong to the same class or have similar features ("birds of a feather flock together") [21]. For example, friends are likely to have similar political beliefs or age, and papers tend to cite papers from the same research area [23]. GNNs *model the homophily principle* by propagating features and aggregating them within various graph neighborhoods via different mechanisms (e.g., averaging, LSTM) [17, 11, 36]. However, in the real world, there are also settings where "opposites attract", leading to networks with *heterophily*: linked nodes are likely from different classes or have dissimilar features. For instance, the majority of people tend to connect with people of the opposite gender in dating networks, different amino acid types are more likely to connect in protein structures, fraudsters are more likely to connect to accomplices than to other fraudsters in online purchasing networks [24].

异质吸引

Since many existing GNNs assume strong homophily, they fail to generalize to networks with heterophily (or low/medium level of homophily). In such cases, we find that even models that ignore the graph structure altogether, such as multilayer perceptrons or MLPs, can outperform a number of existing GNNs. Motivated by this limitation, we make the following contributions:

- **Current Limitations**: We reveal the limitation of GNNs to learn over networks with heterophily, which is ignored in the literature due to evaluation on few benchmarks with similar properties. § 3
- **Key Designs for Heterophily & New Model:** We identify a set of key designs that can boost learning from the graph structure in heterophily without trading off accuracy in homophily: (D1) ego- and neighbor-embedding separation, (D2) higher-order neighborhoods, and (D3) combination of intermediate representations. We justify the designs theoretically, and combine them into a model, H$_2$GCN, that effectively adapts to both heterophily and homophily. We compare it to prior GNN models, and make our code and data available at `https://github.com/GemsLab/H2GCN`. § 3-4
- **Extensive Empirical Evaluation**: We empirically analyze our model and competitive existing GNN models on both synthetic and real networks covering the full spectrum of low-to-high homophily (besides the typically-used benchmarks with strong homophily only). In synthetic networks, our detailed ablation study of H$_2$GCN (which is free of confounding designs) shows that the identified designs result in up to 40% performance gain in heterophily. In real networks, we observe that GNN models utilizing even a subset of our identified designs outperform popular models without them by up to 27% in heterophily, while being competitive in homophily. § 5

2 Notation and Preliminaries

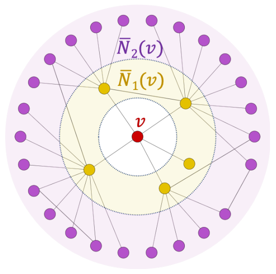


Figure 1: Neighborhoods.

We summarize our notation in Table A.1 (App. A). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected, unweighted graph with nodeset $\mathcal{V}$ and edgeset $\mathcal{E}$. We denote a general neighborhood centered around v as $N(v)$ ($\mathcal{G}$ may have self-loops), the corresponding neighborhood that does *not* include the ego (node v) as $\bar{N}(v)$, and the general neighbors of node v at exactly i hops/steps away (minimum distance) as $N_i(v)$. For example, $N_1(v) = \{u : (u, v) \in \mathcal{E}\}$ are the immediate neighbors of v. Other examples are shown in Fig. 1. We represent the graph by its adjacency matrix $\mathbf{A} \in \{0,1\}^{n\times n}$ and its node feature matrix $\mathbf{X} \in \mathbb{R}^{n\times F}$, where the vector $\mathbf{x}_v$ corresponds to the *ego-feature* of node v, and $\{\mathbf{x}_u : u \in \bar{N}(v)\}$ to its *neighbor-features*.

We further assume a class label vector $\mathbf{y}$, which for each node v contains a unique class label y_v. The goal of semi-supervised node classification is to learn a mapping $\ell : \mathcal{V} \to \mathcal{Y}$, where $\mathcal{Y}$ is the set of labels, given a set of labeled nodes $\mathcal{T}_\mathcal{V} = \{(v_1, y_1), (v_2, y_2), ...\}$ as training data.

Graph neural networks From a probabilistic perspective, most GNN models assume the following local Markov property on node features: for each node $v \in \mathcal{V}$, there exists a neighborhood $N(v)$ such that y_v only depends on the ego-feature $\mathbf{x}_v$ and neighbor-features $\{\mathbf{x}_u : u \in N(v)\}$. Most models derive the class label y_v via the following representation learning approach:

$$\mathbf{r}_v^{(k)} = f\left(\mathbf{r}_v^{(k-1)}, \{\mathbf{r}_u^{(k-1)} : u \in N(v)\}\right), \ \mathbf{r}_v^{(0)} = \mathbf{x}_v, \text{ and } y_v = \arg\max\{\text{softmax}(\mathbf{r}_v^{(K)})\mathbf{W}\}, \qquad (1)$$

where the embedding function f is applied repeatedly in K total rounds, node v's representation (or hidden state vector) at round k, $\mathbf{r}_v^{(k)}$, is learned from its ego- and neighbor-representations in the previous round, and a softmax classifier with learnable weight matrix $\mathbf{W}$ is applied to the final representation of v. Most existing models differ in their definitions of neighborhoods $N(v)$ and embedding function f. A typical definition of neighborhood is $N_1(v)$—i.e., the 1-hop neighbors of v. As for f, in graph convolutional networks (GCN) [17] each node repeatedly averages its own features and those of its neighbors to update its own feature representation. Using an attention mechanism, GAT [36] models the influence of different neighbors more precisely as a weighted average of the ego- and neighbor-features. GraphSAGE [11] generalizes the aggregation beyond averaging, and models the ego-features distinctly from the neighbor-features in its subsampled neighborhood.

Homophily and heterophily In this work, we focus on heterophily in class labels. We first define the edge homophily ratio h as a measure of the graph homophily level, and use it to define graphs with strong homophily/heterophily:

Definition 1 *The edge homophily ratio* $h = \frac{|\{(u,v):(u,v)\in\mathcal{E}\wedge y_u = y_v\}|}{|\mathcal{E}|}$ *is the fraction of edges in a graph which connect nodes that have the same class label (i.e., intra-class edges).*

Definition 2 *Graphs with strong homophily have high edge homophily ratio* $h \to 1$*, while graphs with strong heterophily (i.e., low/weak homophily) have small edge homophily ratio* $h \to 0$.

The edge homophily ratio in Dfn. 1 gives an *overall trend* for all the edges in the graph. The actual level of homophily may vary within different pairs of node classes, i.e., there is different tendency of connection between each pair of classes. In App. B, we give more details about capturing these more complex network characteristics via an empirical *class compatibility matrix* $\mathbf{H}$, whose i, j-th entry is the fraction of outgoing edges to nodes in class j among all outgoing edges from nodes in class i.

[ˌhɛtərədʒɪˈniəti]

Heterophily ≠ Heterogeneity. We remark that heterophily, which we study in this work, is a distinct network concept from heterogeneity. Formally, a network is heterogeneous [34] if it has at least two types of nodes and different relationships between them (e.g., knowledge graphs), and homogeneous if it has a single type of nodes (e.g., users) and a single type of edges (e.g., friendship). The type of nodes in heterogeneous graphs does *not* necessarily match the class labels y_v, therefore both homogeneous and heterogeneous networks may have different levels of homophily.

3 Learning Over Networks with Heterophily

While many GNN models have been proposed, most of them are designed under the assumption of homophily, and are not capable of handling heterophily. As a motivating example, Table 1 shows the mean classification accuracy for several leading GNN models on our synthetic benchmark `syn-cora`, where we can control the homophily/heterophily level (see App. G for details on the data and setup). Here we consider two homophily ratios, $h = 0.1$ and $h = 0.7$, one for high heterophily and one for high homophily. We observe that for heterophily ($h = 0.1$) all existing methods fail to perform better than a Multilayer Perceptron (MLP) with 1 hidden layer, a graph-agnostic baseline that relies solely on the node features for classification (differences in accuracy of MLP for different h are due to randomness). Especially, GCN [17] and GAT [36] show up to 42% worse performance than MLP, highlighting that methods that work well under high homophily ($h = 0.7$) may not be appropriate for networks with low/medium homophily.

Table 1: Example of a heterophily setting ($h = 0.1$) where existing GNNs fail to generalize, and a typical homophily setting ($h = 0.7$): mean accuracy and standard deviation over three runs (cf. App. G).

	$h = \mathbf{0.1}$	$h = \mathbf{0.7}$
GCN [17]	37.14±4.60	84.52±0.54
GAT [36]	33.11±1.20	84.03±0.97
GCN-Cheby [7]	68.10±1.75	84.92±1.03
GraphSAGE [11]	72.89±2.42	85.06±0.51
MixHop [1]	58.93±2.84	84.43±0.94
MLP	74.85±0.76	71.72±0.62
H$_2$GCN **(ours)**	**76.87**±0.43	**88.28**±0.66

Motivated by this limitation, in the following subsections, we discuss and theoretically justify a set of key design choices that, when appropriately incorporated in a GNN framework, can improve the performance in the challenging heterophily settings. Then, we present H$_2$GCN, a model that, thanks to these designs, adapts well to both homophily and heterophily (Table 1, last row). In Section 5, we provide a comprehensive empirical analysis on both synthetic and real data with varying homophily levels, and show that the identified designs significantly improve the performance of GNNs (not limited to H$_2$GCN) by effectively leveraging the graph structure in challenging heterophily settings, while maintaining competitive performance in homophily.

3.1 Effective Designs for Networks with Heterophily

We have identified three key designs that—when appropriately integrated—can help improve the performance of GNN models in heterophily settings: (D1) ego- and neighbor-embedding separation; (D2) higher-order neighborhoods; and (D3) combination of intermediate representations. While these designs have been utilized separately in some prior works [11, 7, 1, 38], we are the first to discuss their importance *under heterophily* by providing novel theoretical justifications and an extensive empirical analysis on a variety of datasets.

3.1.1 (D1) Ego- and Neighbor-embedding Separation

The first design entails encoding each ego-embedding (i.e., a node's embedding) *separately* from the aggregated embeddings of its neighbors, since they are likely to be dissimilar in heterophily settings. Formally, the representation (or hidden state vector) learned for each node v at round k is given as:

$$\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\mathbf{r}_v^{(k-1)}, \texttt{AGGR}(\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}(v)\})\right), \tag{2}$$

the neighborhood $\bar{N}(v)$ does *not* include v (no self-loops), the `AGGR` function aggregates representations *only* from the neighbors (in some way—e.g., average), and `AGGR` and `COMBINE` may be followed by a non-linear transformation. For heterophily, after aggregating the neighbors' representations, the definition of COMBINE (akin to 'skip connection' between layers) is critical: a simple way to combine the ego- and the aggregated neighbor-embeddings without 'mixing' them is with concatenation as in GraphSAGE [11]—rather than averaging *all* of them as in the GCN model by Kipf and Welling [17].

Intuition. In heterophily settings, by definition (Dfn. 2), the class label y_v and original features $\mathbf{x}_v$ of a node and those of its neighboring nodes $\{(y_u, \mathbf{x}_u) : u \in \bar{N}(v)\}$ (esp. the direct neighbors $\bar{N}_1(v)$) may be different. However, the typical GCN design that mixes the embeddings through an average [17] or weighted average [36] as the COMBINE function results in final embeddings that are similar across neighboring nodes (especially within a community or cluster) *for any set of original features* [28]. While this may work well in the case of homophily, where neighbors likely belong to the same cluster and class, it poses severe challenges in the case of heterophily: it is not possible to distinguish neighbors from different classes based on the (similar) learned representations. Choosing a COMBINE function that separates the representations of each node v and its neighbors $\bar{N}(v)$ allows for more expressiveness, where the skipped or non-aggregated representations can evolve separately over multiple rounds of propagation without becoming prohibitively similar.

Theoretical Justification. We prove theoretically that, under some conditions, a GCN layer that co-embeds ego- and neighbor-features is less capable of generalizing to heterophily than a layer that embeds them separately. We measure its generalization ability by its robustness to test/train data deviations. We give the proof of the theorem in App. C.1. Though the theorem applies to specific conditions, our empirical analysis shows that it holds in more general cases (§ 5).

Theorem 1 *Consider a graph $\mathcal{G}$ without self-loops (§ 2) with node features $\mathbf{x}_v = \text{onehot}(y_v)$ for each node v, and an equal number of nodes per class $y \in \mathcal{Y}$ in the training set $\mathcal{T}_\mathcal{V}$. Also assume that all nodes in $\mathcal{T}_\mathcal{V}$ have degree d, and proportion h of their neighbors belong to the same class, while proportion $\frac{1-h}{|\mathcal{Y}|-1}$ of them belong to any other class (uniformly). Then for $h < \frac{1-|\mathcal{Y}|+2d}{2|\mathcal{Y}|d}$, a simple GCN layer formulated as $(\mathbf{A}+\mathbf{I})\mathbf{XW}$ is less robust, i.e., misclassifies a node for smaller train/test data deviations, than a $\mathbf{AXW}$ layer that separates the ego- and neighbor-embeddings.*

Observations. In Table 1, we observe that GCN, GAT, and MixHop, which 'mix' the ego- and neighbor-embeddings explicitly[1], perform poorly in the heterophily setting. On the other hand, GraphSAGE that separates the embeddings (e.g., it concatenates the two embeddings and then applies a non-linear transformation) achieves 33-40% better performance in this setting.

3.1.2 (D2) Higher-order Neighborhoods

The second design involves explicitly aggregating information from higher-order neighborhoods in each round k, beyond the immediate neighbors of each node:

$$\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\mathbf{r}_v^{(k-1)},\ \texttt{AGGR}(\{\mathbf{r}_u^{(k-1)} : u \in N_1(v)\}),\ \texttt{AGGR}(\{\mathbf{r}_u^{(k-1)} : u \in N_2(v)\}), \ldots\right) \tag{3}$$

where $N_i(v)$ denotes the neighbors of v at *exactly* i hops away, and the AGGR functions applied to different neighborhoods can be the same or different. This design—employed in GCN-Cheby [7] and MixHop [1]—augments the *implicit* aggregation over higher-order neighborhoods that most GNN models achieve through multiple rounds of first-order propagation based on variants of Eq. (2).

Intuition. To show why higher-order neighborhoods help in the heterophily settings, we first define *homophily-dominant* and *heterophily-dominant* neighborhoods:

Definition 3 *$N(v)$ is expectedly homophily-dominant if $P(y_u = y_v | y_v) \geq P(y_u = y | y_v), \forall u \in N(v)$ and $y \in \mathcal{Y} \neq y_v$. If the opposite inequality holds, $N(v)$ is expectedly heterophily-dominant.*

图的性质与
模型消息传导机制的
综合作用结果

From this definition, we can see that expectedly homophily-dominant neighborhoods are more beneficial for GNN layers, as in such neighborhoods the class label y_v of each node v can *in expectation* be determined by the majority of the class labels in $N(v)$. In the case of heterophily, we have seen empirically that although the immediate neighborhoods may be heterophily-dominant, the higher-order neighborhoods may be homophily-dominant and thus provide more relevant context. This observation is also confirmed by recent works [2, 6] in the context of binary attribute prediction.

[1] These models consider self-loops, which turn each ego also into a neighbor, and thus mix the ego- and neighbor-representations. E.g., GCN and MixHop operate on the symmetric normalized adjacency matrix augmented with self-loops: $\hat{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\hat{\mathbf{D}}^{-\frac{1}{2}}$, where $\mathbf{I}$ is the identity and $\hat{\mathbf{D}}$ the degree matrix of $\mathbf{A}+\mathbf{I}$.

Theoretical Justification. Below we formalize the above observation for 2-hop neighborhoods under non-binary attributes (labels), and prove one case when they are homophily-dominant in App. C.2:

Theorem 2 *Consider a graph $\mathcal{G}$ without self-loops (§ 2) with label set $\mathcal{Y}$, where for each node v, its neighbors' class labels $\{y_u : u \in N(v)\}$ are conditionally independent given y_v, and $P(y_u = y_v|y_v) = h$, $P(y_u = y|y_v) = \frac{1-h}{|\mathcal{Y}|-1}, \forall y \neq y_v$. Then, the 2-hop neighborhood $N_2(v)$ for a node v will always be homophily-dominant in expectation.*

Observations. Under heterophily ($h = 0.1$), GCN-Cheby, which models different neighborhoods by combining Chebyshev polynomials to approximate a higher-order graph convolution operation [7], outperforms GCN and GAT, which aggregate over only the immediate neighbors N_1, by up to +31% (Table 1). MixHop, which explicitly models 1-hop and 2-hop neighborhoods (though 'mixes' the ego- and neighbor-embeddings[1], violating design D1), also outperforms these two models.

3.1.3 (D3) Combination of Intermediate Representations

The third design combines the intermediate representations of each node at the final layer:

$$\mathbf{r}_v^{(\text{final})} = \texttt{COMBINE}\left(\mathbf{r}_v^{(1)}, \mathbf{r}_v^{(2)}, \ldots, \mathbf{r}_v^{(K)}\right) \tag{4}$$

to explicitly capture local *and* global information via COMBINE functions that leverage each representation separately–e.g., concatenation, LSTM-attention [38]. This design is introduced in jumping knowledge networks [38] and shown to increase the representation power of GCNs under *homophily*.

Intuition. Intuitively, each round collects information with different locality—earlier rounds are more local, while later rounds capture increasingly more global information (implicitly, via propagation). Similar to D2 (which models explicit neighborhoods), this design models the distribution of neighbor representations in low-homophily networks more accurately. It also allows the class prediction to leverage different neighborhood ranges in different networks, adapting to their structural properties.

Theoretical Justification. The benefit of combining intermediate representations can be theoretically explained from the spectral perspective. Assuming a GCN-style layer—where propagation can be viewed as spectral filtering—, the higher order polynomials of the normalized adjacency matrix $\mathbf{A}$ is a low-pass filter [37], so intermediate outputs from earlier rounds contain higher-frequency components than outputs from later rounds. At the same time, the following theorem holds for graphs with heterophily, where we view class labels as graph signals (as in graph signal processing):

Theorem 3 *Consider graph signals (label vectors) $\mathbf{s}, \mathbf{t} \in \{0,1\}^{|\mathcal{V}|}$ defined on an undirected graph $\mathcal{G}$ with edge homophily ratios h_s and h_t, respectively. If $h_s < h_t$, then signal $\mathbf{s}$ has higher energy (Dfn. 5) in high-frequency components than $\mathbf{t}$ in the spectrum of unnormalized graph Laplacian $\mathbf{L}$.*

In other words, in heterophily settings, the label distribution contains more information at higher than lower frequencies (see proof in App. C.3). Thus, by combining the intermediate outputs from different layers, this design captures both low- and high-frequency components in the final representation, which is critical in heterophily settings, and allows for more expressiveness in the general setting.

Observations. By concatenating the intermediate representations from two rounds with the embedded ego-representation (following the jumping knowledge framework [38]), GCN's accuracy increases to $58.93\%{\pm}3.17$ for $h = 0.1$, a 20% improvement over its counterpart without design D3 (Table 1).

本地/邻居
不同层次邻居
不同层次本地

Summary of designs To sum up, D1 models (at each layer) the ego- and neighbor-representations *distinctly*, D2 leverages (at each layer) representations of neighbors at different distances *distinctly*, and D3 leverages (at the final layer) the learned ego-representations at previous layers *distinctly*.

3.2 H$_2$GCN: A Framework for Networks with Homophily or Heterophily

We now describe H$_2$GCN, which exemplifies how effectively combining designs D1-D3 can help better adapt to the whole spectrum of low-to-high homophily, while avoiding interference with other designs. It has three stages (Alg. 1, App. D): **(S1)** feature embedding, **(S2)** neighborhood aggregation, and **(S3)** classification.

The *feature embedding stage* **(S1)** uses a graph-agnostic dense layer to generate for each node v the feature embedding $\mathbf{r}_v^{(0)} \in \mathbb{R}^p$ based on its ego-feature $\mathbf{x}_v$: $\mathbf{r}_v^{(0)} = \sigma(\mathbf{x}_v \mathbf{W}_e)$, where σ is an optional non-linear function, and $\mathbf{W}_e \in \mathbb{R}^{F \times p}$ is a learnable weight matrix.

In the *neighborhood aggregation stage* **(S2)**, the generated embeddings are aggregated and repeatedly updated within the node's neighborhood for K rounds. Following designs D1 and D2, the neighborhood $N(v)$ of our framework involves two sub-neighborhoods without the egos: the 1-hop graph neighbors $\bar{N}_1(v)$ and the 2-hop neighbors $\bar{N}_2(v)$, as shown in Fig. 1:

$$\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\texttt{AGGR}\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}_1(v)\}, \texttt{AGGR}\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}_2(v)\}\right). \tag{5}$$

We set `COMBINE` as concatenation (as to not mix different neighborhood ranges), and `AGGR` as a degree-normalized average of the neighbor-embeddings in sub-neighborhood $\bar{N}_i(v)$:

$$\mathbf{r}_v^{(k)} = \left(\mathbf{r}_{v,1}^{(k)} \| \mathbf{r}_{v,2}^{(k)}\right) \quad \text{and} \quad \mathbf{r}_{v,i}^{(k)} = \texttt{AGGR}\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}_i(v)\} = \textstyle\sum_{u \in \bar{N}_i(v)} \mathbf{r}_u^{(k-1)} d_{v,i}^{-1/2} d_{u,i}^{-1/2}, \tag{6}$$

where $d_{v,i} = |\bar{N}_i(v)|$ is the i-hop degree of node v (i.e., number of nodes in its i-hop neighborhood). Unlike Eq. (2), here we do *not* combine the ego-embedding of node v with the neighbor-embeddings. We found that removing the usual nonlinear transformations per round, as in SGC [37], works better (App. D.2), in which case we only need to include the ego-embedding in the final representation. By design D3, each node's final representation combines all its intermediate representations:

$$\mathbf{r}_v^{(\text{final})} = \texttt{COMBINE}\left(\mathbf{r}_v^{(0)}, \mathbf{r}_v^{(1)}, \ldots, \mathbf{r}_v^{(K)}\right), \tag{7}$$

where we empirically find concatenation works better than max-pooling [38] as the `COMBINE` function.

In the *classification stage* **(S3)**, the node is classified based on its final embedding $\mathbf{r}_v^{(\text{final})}$:

$$y_v = \arg\max\{\text{softmax}(\mathbf{r}_v^{(\text{final})} \mathbf{W}_c)\}, \tag{8}$$

where $\mathbf{W}_c \in \mathbb{R}^{(2^{K+1}-1)p \times |\mathcal{Y}|}$ is a learnable weight matrix. We visualize our framework in App. D.

Time complexity The feature embedding stage **(S1)** takes $\mathrm{O}(\mathrm{nnz}(\mathbf{X})\, p)$, where $\mathrm{nnz}(\mathbf{X})$ is the number of non-0s in feature matrix $\mathbf{X} \in \mathbb{R}^{n \times F}$, and p is the dimension of the feature embeddings. The neighborhood aggregation stage **(S2)** takes $\mathrm{O}\left(|\mathcal{E}| d_{\max}\right)$ to derive the 2-hop neighborhoods via sparse-matrix multiplications, where $d_{\max}$ is the maximum degree of all nodes, and $\mathrm{O}\left(2^K(|\mathcal{E}| + |\mathcal{E}_2|)p\right)$ for K rounds of aggregation, where $|\mathcal{E}_2| = \frac{1}{2}\sum_{v \in \mathcal{V}} |\bar{N}_2(v)|$. We give a detailed analysis in App. D.

4 Other Related Work

We discuss relevant work on GNNs here, and give other related work (e.g., classification under heterophily) in Appendix E. Besides the models mentioned above, there are various comprehensive reviews describing previously proposed architectures [42, 5, 41]. Recent work has investigated GNN's ability to capture graph information, proposing diagnostic measurements based on feature smoothness and label smoothness [12] that may guide the learning process. To capture more graph information, other works generalize graph convolution outside of immediate neighborhoods. For example, apart from MixHop [1] (cf. § 3.1), Graph Diffusion Convolution [18] replaces the adjacency matrix with a sparsified version of a diffusion matrix (e.g., heat kernel or PageRank). Geom-GCN [26] precomputes unsupervised node embeddings and uses neighborhoods defined by geometric relationships in the resulting latent space to define graph convolution. Some of these works [1, 26, 12] acknowledge the challenges of learning from graphs with heterophily. Others have noted that node labels may have complex relationships that should be modeled directly. For instance, Graph Agreement Models [33] augment the classification task with an agreement task, co-training a model to predict whether pairs of nodes share the same label; Graph Markov Neural Networks [27] model the joint label distribution with a conditional random field, trained with expectation maximization using GNNs; Correlated Graph Neural Networks [15] model the correlation structure in the residuals of a regression task with a multivariate Gaussian, and can learn negative label correlations for neighbors in heterophily (for binary class labels); and the recent CPGNN [43] method models more complex label correlations by integrating the compatibility matrix notion from belief propagation [10] into GNNs.

Comparison of H$_2$GCN to existing GNN models As shown in Table 2, H$_2$GCN differs from existing GNN models with respect to designs D1-D3, and their implementations (we give more details in App. D). Notably, H$_2$GCN learns a graph-agnostic feature embedding in stage **(S1)**, and skips the non-linear embeddings of aggregated representations per round that other models use (e.g., GraphSAGE, MixHop, GCN), resulting in a simpler yet powerful architecture.

Table 2: Design Comparison.

Method	**D1**	**D2**	**D3**
GCN [17]	✗	✗	✗
GAT [36]	✗	✗	✗
GCN-Cheby [7]	✗	✓	✗
GraphSAGE [11]	✓	✗	✗
MixHop [1]	✗	✓	✗
H$_2$GCN (proposed)	✓	✓	✓

Table 3: Statistics for Synthetic Datasets

Benchmark Name	#Nodes $\lvert\mathcal{V}\rvert$	#Edges $\lvert\mathcal{E}\rvert$	#Classes $\lvert\mathcal{Y}\rvert$	#Features F	Homophily h	#Graphs
`syn-cora`	$1,490$	$2,965$ to $2,968$	5	`cora` [30, 39]	$[0, 0.1, \ldots, 1]$	33 (3 per h)
`syn-products`	$10,000$	$59,640$ to $59,648$	10	`ogbn-products` [13]	$[0, 0.1, \ldots, 1]$	33 (3 per h)

5 Empirical Evaluation

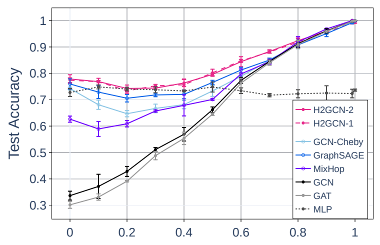
We show the significance of designs D1-D3 on synthetic and real graphs with low-to-high homophily (Tab. 3, 5) via an ablation study of H$_2$GCN and comparison of models with and without the designs.

Baseline models We consider MLP with 1 hidden layer, and all the methods listed in Table 2. For H$_2$GCN, we model the first- and second-order neighborhoods ($\bar{N}_1$ and $\bar{N}_2$), and consider two variants: H$_2$GCN-1 uses one embedding round ($K = 1$) and H$_2$GCN-2 uses two rounds ($K = 2$). We tune all the models on the same train/validation splits (see App. F for details).

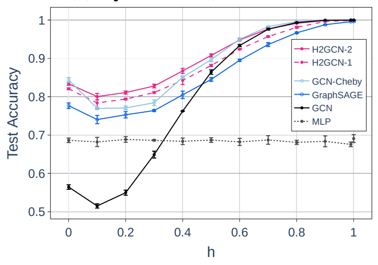
5.1 Evaluation on Synthetic Benchmarks

Synthetic datasets & setup We generate synthetic graphs with various homophily ratios h (Tab. 3) by adopting an approach similar to [16]. In App. G, we describe the data generation process, the experimental setup, and the data statistics in detail. All methods share the same training, validation and test splits (25%, 25%, 50% per class), and we report the average accuracy and standard deviation (stdev) over three generated graphs per heterophily level and benchmark dataset.

Model comparison Figure 2 shows the mean test accuracy (and stdev) over all random splits of our synthetic benchmarks. We observe similar trends on both benchmarks: H$_2$GCN has the best trend overall, outperforming the baseline models in most heterophily settings, while tying with other models in homophily. The performance of GCN, GAT and MixHop, which mix the ego- and neighbor-embeddings, increases with respect to the homophily level. But, while they achieve near-perfect accuracy under strong homophily ($h \to 1$), they are significantly less accurate than MLP (near-flat performance curve as it is graph-agnostic) for many heterophily settings. GraphSAGE and GCN-Cheby, which leverage some of the identified designs D1-D3 (Table 2, § 3), are more competitive in such settings. We note that all the methods—except GCN and GAT—learn more effectively under perfect heterophily (h=0) than weaker settings (e.g., $h \in [0.1, 0.3]$), as evidenced by the J-shaped performance curves in low-homophily ranges.



(a) `syn-cora` (Table G.2)



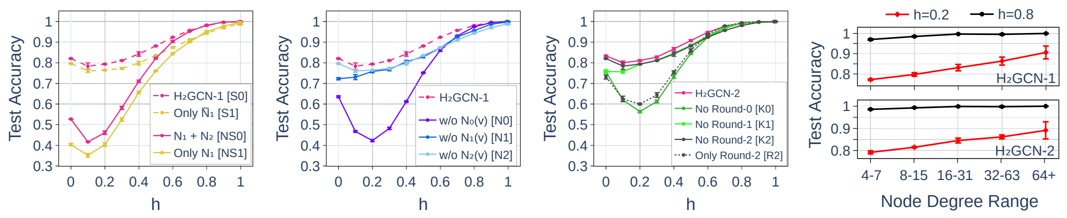
(b) `syn-products` (Table G.3). MixHop acc $< 30\%$; GAT acc $< 50\%$ for $h < 0.4$.

Figure 2: Performance of GNN models on synthetic datasets. H$_2$GCN-2 outperforms baseline models in most heterophily settings, while tying with other models in homophily.

Significance of design choices Using `syn-products`, we show the significance of designs D1-D3 (§ 3.1) through ablation studies with variants of H$_2$GCN (Fig. 3, Table G.4).

(D1) *Ego- and Neighbor-embedding Separation.* We consider H$_2$GCN-1 variants that *separate* the ego- and neighbor-embeddings and model: (`S0`) neighborhoods $\bar{N}_1$ and $\bar{N}_2$ (i.e., H$_2$GCN-1); (`S1`) only the 1-hop neighborhood $\bar{N}_1$ in Eq. (5); and their counterparts that do *not separate* the two embeddings and use: (`NS0`) neighborhoods N_1 and N_2 (including v); and (`NS1`) only the 1-hop neighborhood N_1. Figure 3a shows that the variants that learn separate embedding functions significantly outperform the others (`NS0/1`) in heterophily settings ($h < 0.7$) by up to 40%, which shows that design D1 is critical for success in heterophily. H$_2$GCN-1 (`S0`) performs best in homophily.

(D2) *Higher-order Neighborhoods.* For this design, we consider three variants of H$_2$GCN-1 without specific neighborhoods: (`N0`) without the 0-hop neighborhood $N_0(v) = v$ (i.e, the ego-embedding) (`N1`) without $\bar{N}_1(v)$; and (`N2`) without $\bar{N}_2(v)$. Figure 3b shows that H$_2$GCN-1 consistently performs better than all the variants, indicating that combining all sub-neighborhoods works best. Among the variants, in heterophily settings, $N_0(v)$ contributes most to the performance (`N0` causes significant decrease in accuracy), followed by $\bar{N}_1(v)$, and $\bar{N}_2(v)$. However, when $h \geq 0.7$, the importance of sub-neighborhoods is reversed. Thus, the ego-features are the most important in heterophily, and



(a) Design D1: Embedding separation. (b) Design D2: Higher-order neighborhoods. (c) Design D3: Intermediate representations. (d) Accuracy per degree in hetero/homo-phily.

Figure 3: (a)-(c): Significance of design choices D1-D3 via ablation studies. (d): Performance of H_2GCN for different node degree ranges. In heterophily, the performance gap between low- and high-degree nodes is significantly larger than in homophily, i.e., low-degree nodes pose challenges.

higher-order neighborhoods contribute the most in homophily. The design of H_2GCN allows it to effectively combine information from different neighborhoods, adapting to all levels of homophily.

(D3) *Combination of Intermediate Representations.* We consider three variants (`K-0,1,2`) of H_2GCN-2 that drop from the final representation of Eq. (7) the 0^{th}, 1^{st} or 2^{nd}-round intermediate representation, respectively. We also consider only the 2^{nd} intermediate representation as final, which is akin to what the other GNN models do. Figure 3c shows that H_2GCN-2, which combines all the intermediate representations, performs the best, followed by the variant `K2` that skips the round-2 representation. The ego-embedding is the most important for heterophily $h \leq 0.5$ (see trend of `K0`).

The challenging case of low-degree nodes Figure 3d plots the mean accuracy of H_2GCN variants on `syn-products` for different node degree ranges both in a heterophily and a homophily setting ($h \in \{0.2, 0.8\}$). We observe that under heterophily there is a significantly bigger performance gap between low- and high-degree nodes: 13% for H_2GCN-1 (10% for H_2GCN-2) vs. less than 3% under homophily. This is likely due to the importance of the *distribution* of class labels in each neighborhood under heterophily, which is harder to estimate accurately for low-degree nodes with few neighbors. On the other hand, in homophily, neighbors are likely to have similar classes $y \in \mathcal{Y}$, so the neighborhood size does not have as significant impact on the accuracy.

5.2 Evaluation on Real Benchmarks

Real datasets & setup We now evaluate the performance of our model and existing GNNs on a variety of real-world datasets [35, 29, 30, 22, 4, 31] with edge homophily ratio h ranging from strong heterophily to strong homophily, going beyond the traditional Cora, Pubmed and Citeseer graphs that have strong homophily (hence the good performance of existing GNNs on them). We summarize the data in Table 5, and describe them in App. H, where we also point out potential data limitations. For all benchmarks (except `Cora-Full`), we use the feature vectors, class labels, and 10 random splits (48%/32%/20% of nodes per class for train/validation/test[2]) provided by [26]. For Cora-Full, we generate 3 random splits, with 25%/25%/50% of nodes per class for train/validation/test.

Table 4: Real benchmarks: Average rank per method (and their employed designs among D1-D3) under heterophily (benchmarks with $h \leq 0.3$), homophily ($h \geq 0.7$), and across the full spectrum ("Overall"). The "*" denotes ranks based on results reported in [26].

Method (Designs)	Het.	Hom.	Overall
H_2GCN-1 (D1, D2, D3)	3.8	3.0	3.6
H_2GCN-2 (D1, D2, D3)	4.0	2.0	3.3
GraphSAGE (D1)	5.0	6.0	5.3
GCN-Cheby (D2)	7.0	6.3	6.8
MixHop (D2)	6.5	6.0	6.3
GraphSAGE+JK (D1, D3)	5.0	7.0	5.7
GCN-Cheby+JK (D2, D3)	3.7	7.7	5.0
GCN+JK (D3)	7.2	8.7	7.7
GCN	9.8	5.3	8.3
GAT	11.5	10.7	11.2
GEOM-GCN*	8.2	4.0	6.8
MLP	6.2	11.3	7.9

Effectiveness of design choices Table 4 gives the average ranks of our H_2GCN variants and other models on real benchmarks with heterophily, homophily, and across the full spectrum. Table 5 gives detailed results (mean accuracy and stdev) per benchmark. We observe that models which utilize all or subsets of our identified designs D1-D3 (§ 3.1) perform significantly better than GCN and GAT which lack these designs, especially in heterophily. Next, we discuss the effectiveness of each design.

(D1) *Ego- and Neighbor-embedding Separation.* We compare GraphSAGE, which *separates* the ego- and neighbor-embeddings, and GCN that does not. In heterophily settings, GraphSAGE has

[2][26] claims that the ratios are 60%/20%/20%, which is different from the actual data splits shared on GitHub.

Table 5: Real data: mean accuracy ± stdev over different data splits. Best model per benchmark highlighted in gray. The "*" results are obtained from [26] and "N/A" denotes non-reported results.

	Texas	**Wisconsin**	**Actor**	**Squirrel**	**Chameleon**	**Cornell**	**Cora Full**	**Citeseer**	**Pubmed**	**Cora**
Hom. ratio h	**0.11**	**0.21**	**0.22**	**0.22**	**0.23**	**0.3**	**0.57**	**0.74**	**0.8**	**0.81**
#Nodes $\lvert\mathcal{V}\rvert$	183	251	7,600	5,201	2,277	183	19,793	3,327	19,717	2,708
#Edges $\lvert\mathcal{E}\rvert$	295	466	26,752	198,493	31,421	280	63,421	4,676	44,327	5,278
#Classes $\lvert\mathcal{Y}\rvert$	5	5	5	5	5	5	70	7	3	6
H_2GCN-1	84.86±6.77	86.67±4.69	35.86±1.03	36.42±1.89	57.11±1.58	82.16±4.80	68.13±0.49	77.07±1.64	89.40±0.34	86.92±1.37
H_2GCN-2	82.16±5.28	85.88±4.22	35.62±1.30	37.90±2.02	59.39±1.98	82.16±6.00	69.05±0.37	76.88±1.77	89.59±0.33	87.81±1.35
GraphSAGE	82.43±6.14	81.18±5.56	34.23±0.99	41.61±0.74	58.73±1.68	75.95±5.01	65.14±0.75	76.04±1.30	88.45±0.50	86.90±1.04
GCN-Cheby	77.30±4.07	79.41±4.46	34.11±1.09	43.86±1.64	55.24±2.76	74.32±7.46	67.41±0.69	75.82±1.53	88.72±0.55	86.76±0.95
MixHop	77.84±7.73	75.88±4.90	32.22±2.34	43.80±1.48	60.50±2.53	73.51±6.34	65.59±0.34	76.26±1.33	85.31±0.61	87.61±0.85
GraphSAGE+JK	83.78±2.21	81.96±4.96	34.28±1.01	40.85±1.29	58.11±1.97	75.68±4.03	65.31±0.58	76.05±1.37	88.34±0.62	85.96±0.83
Cheby+JK	78.38±6.37	82.55±4.57	35.14±1.37	45.03±1.73	63.79±2.27	74.59±7.87	66.87±0.29	74.98±1.18	89.07±0.30	85.49±1.27
GCN+JK	66.49±6.64	74.31±6.43	34.18±0.85	40.45±1.61	63.42±2.00	64.59±8.68	66.72±0.61	74.51±1.75	88.41±0.45	85.79±0.92
GCN	59.46±5.25	59.80±6.99	30.26±0.79	36.89±1.34	59.82±2.58	57.03±4.67	68.39±0.32	76.68±1.64	87.38±0.66	87.28±1.26
GAT	58.38±4.45	55.29±8.71	26.28±1.73	30.62±2.11	54.69±1.95	58.92±3.32	59.81±0.92	75.46±1.72	84.68±0.44	82.68±1.80
GEOM-GCN*	67.57	64.12	31.63	38.14	60.90	60.81	N/A	77.99	90.05	85.27
MLP	81.89±4.78	85.29±3.61	35.76±0.98	29.68±1.81	46.36±2.52	81.08±6.37	58.76±0.50	72.41±2.18	86.65±0.35	74.75±2.22

an average rank of 5.0 compared to 9.8 for GCN, and outperforms GCN in almost all heterophily benchmarks by up to 23%. In homophily settings ($h \geq 0.7$), GraphSAGE ranks close to GCN (6.0 vs. 5.3), and GCN never outperforms GraphSAGE by more than 1% in mean accuracy. These results support the importance of D1 for success in heterophily and comparable performance in homophily.

(D2) *Higher-order Neighborhoods.* To show the benefits of design D2 under heterophily, we compare the performance of GCN-Cheby and MixHop—which define higher-order graph convolutions—to that of (first-order) GCN. Under heterophily, GCN-Cheby (rank 7.0) and MixHop (rank 6.5) have better performance than GCN (rank 9.8), and outperform the latter in all but one heterophily benchmarks by up to 20%. In most homophily benchmarks, the performance difference between these methods is less than 1%. Our observations highlight the importance of D2, especially in heterophily.

(D3) *Combination of Intermediate Representations.* We compare GraphSAGE, GCN-Cheby and GCN to their corresponding variants enhanced with JK connections [38]. GCN and GCN-Cheby benefit significantly from D3 in heterophily: their average ranks improve (9.8 vs. 7.2 and 7 vs 3.7, respectively) and their mean accuracies increase by up to 14% and 8%, respectively, in heterophily benchmarks. Though GraphSAGE+JK performs better than GraphSAGE on half of the heterophily benchmarks, its average rank remains unchanged. This may be due to the marginal benefit of D3 when combined with D1, which GraphSAGE employs. Under homophily, the performance with and without JK connections is similar (gaps mostly less than 2%), matching the observations in [38].

While other design choices and implementation details may confound a comparative evaluation of D1-D3 in different models (motivating our introduction of H_2GCN and our ablation study in § 3.1), these observations support the effectiveness of our identified designs on diverse GNN architectures and real-world datasets, and affirm our findings in the ablation study. We also observe that our H_2GCN variants, which combine the three identified designs, have consistently strong performance across the *full spectrum* of low-to-high homophily: H_2GCN-2 achieves the best average rank (3.3) across all datasets (or homophily ratios h), followed by H_2GCN-1 (3.6).

Additional model comparison In Table 4, we also report the *best* results among the *three* recently-proposed GEOM-GCN variants (§ 4), directly from the paper [26]: other models (including ours) outperform this method significantly under heterophily. We note that MLP is a competitive baseline under heterophily (ranked 6.2), indicating that many existing models do not use the graph information effectively, or the latter is misleading in such cases. All models perform poorly on `Squirrel` and `Actor` likely due to their low-quality node features (small correlation with class labels). Also, `Squirrel` and `Chameleon` are dense, with many nodes sharing the same neighbors.

6 Conclusion

We have focused on characterizing the representation power of GNNs in challenging settings with heterophily or low homophily, which is understudied in the literature. We have highlighted the current limitations of GNNs, presented designs that increase representation power under heterophily and are theoretically justified with perturbation analysis and graph signal processing, and introduced the H_2GCN model that adapts to both heterophily and homophily by effectively synthetizing these designs. We analyzed various challenging datasets, going beyond the often-used benchmark datasets (Cora, Pubmed, Citeseer), and leave as future work extending to a larger-scale experimental testbed.

Broader Impact

Homophily and heterophily are not intrinsically ethical or unethical—they are both phenomena existing in the nature, resulting in the popular proverbs "birds of a feather flock together" and "opposites attract". However, many popular GNN models implicitly assume homophily; as a result, if they are applied to networks that do not satisfy the assumption, the results may be biased, unfair, or erroneous. In some applications, the homophily assumption may have ethical implications. For example, a GNN model that intrinsically assumes homophily may contribute to the so-called "filter bubble" phenomenon in a recommendation system (reinforcing existing beliefs/views, and downplaying the opposite ones), or make minority groups less visible in social networks. In other cases, a reliance on homophily may hinder scientific progress. Among other domains, this is critical for applying GNN models to molecular and protein structures, where the connected nodes often belong to different classes, and thus successful methods will need to model heterophily successfully.

Our work has the potential to rectify some of these potential negative consequences of existing GNN work. While our methodology does not change the amount of homophily in a network, moving beyond a reliance on homophily can be a key to improve the fairness, diversity and performance in applications using GNNs. We hope that this paper will raise more awareness and discussions regarding the homophily limitations of existing GNN models, and help researchers design models which have the power of learning in both homophily and heterophily settings.

Acknowledgments and Disclosure of Funding

We thank the reviewers for their constructive feedback. This material is based upon work supported by the National Science Foundation under CAREER Grant No. IIS 1845491 and 1452425, Army Young Investigator Award No. W911NF1810397, an Adobe Digital Experience research faculty award, an Amazon faculty award, a Google faculty award, and AWS Cloud Credits for Research. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Hrayr Harutyunyan, Nazanin Alipourfard, Kristina Lerman, Greg Ver Steeg, and Aram Galstyan. 2019. MixHop: Higher-Order Graph Convolution Architectures via Sparsified Neighborhood Mixing. In *International Conference on Machine Learning (ICML)*.

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

[3] A. L. Barabasi and R. Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (October 1999), 509–512. `http://view.ncbi.nlm.nih.gov/pubmed/10521342`

[4] Aleksandar Bojchevski and Stephan Günnemann. 2018. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *International Conference on Learning Representations (ICLR)*. `https://openreview.net/forum?id=r1ZdKJ-0W`

[5] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. 2020. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. *arXiv preprint arXiv:2005.03675* (2020).

[6] Alex Chin, Yatong Chen, Kristen M. Altenburger, and Johan Ugander. 2019. Decoupled smoothing on graphs. In *Proceedings of the 2019 World Wide Web Conference*. 263–272.

[7] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*. 3844–3852.

[8] Dhivya Eswaran, Stephan Günnemann, Christos Faloutsos, Disha Makhija, and Mohit Kumar. 2017. Zoobp: Belief propagation for heterogeneous networks. *Proceedings of the VLDB Endowment* 10, 5 (2017), 625–636.

[9] Wolfgang Gatterbauer. 2014. Semi-supervised learning with heterophily. *arXiv preprint arXiv:1412.3100* (2014).

[10] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. 2015. Linearized and Single-Pass Belief Propagation. *Proceedings of the VLDB Endowment* 8, 5 (2015).

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in neural information processing systems (NeurIPS)*. 1024–1034.

[12] Yifan Hou, Jian Zhang, James Cheng, Kaili Ma, Richard T. B. Ma, Hongzhi Chen, and Ming-Chang Yang. 2020. Measuring and Improving the Use of Graph Information in Graph Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[13] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

[14] D. Jensen J. Neville. 2000. Iterative classification in relational data, In In Proc. AAAI. *Workshop on Learning Statistical Models from Relational*, 13–20.

[15] Junteng Jia and Austion R Benson. 2020. Residual Correlation in Graph Neural Network Regression. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 588–598.

[16] Fariba Karimi, Mathieu Génois, Claudia Wagner, Philipp Singer, and Markus Strohmaier. 2017. Visibility of minorities in social networks. *arXiv preprint arXiv:1702.00150* (2017).

[17] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[18] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[19] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. 2011. Unifying Guilt-by-Association Approaches: Theorems and Fast Algorithms. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*. 245–260.

[20] Qing Lu and Lise Getoor. 2003. Link-Based Classification. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML)*. AAAI Press, 496–503.

[21] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology* 27, 1 (2001), 415–444.

[22] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. 2012. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, Vol. 8.

[23] Mark Newman. 2018. *Networks*. Oxford university press.

[24] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. 2007. NetProbe: A Fast and Scalable System for Fraud Detection in Online Auction Networks. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 201–210.

[25] Leto Peel. 2017. Graph-based semi-supervised learning for relational networks. In *Proceedings of the 2017 SIAM International Conference on Data Mining*. SIAM, 435–443.

[26] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*. `https://openreview.net/forum?id=S1e2agrFvS`

[27] Meng Qu, Yoshua Bengio, and Jian Tang. 2019. GMNN: Graph Markov Neural Networks. In *International Conference on Machine Learning (ICML)*. 5241–5250.

[28] Ryan A. Rossi, Di Jin, Sungchul Kim, Nesreen Ahmed, Danai Koutra, and John Boaz Lee. 2020. On Proximity and Structural Role-based Embeddings in Networks: Misconceptions, Techniques, and Applications. *ACM Transactions on Knowledge Discovery from Data (TKDD)* (2020).

[29] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2019. Multi-scale attributed node embedding. *arXiv preprint arXiv:1909.13021* (2019).

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

[31] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS 2018* (2018).

[32] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine* 30, 3 (2013), 83–98.

[33] Otilia Stretcu, Krishnamurthy Viswanathan, Dana Movshovitz-Attias, Emmanouil Platanios, Sujith Ravi, and Andrew Tomkins. 2019. Graph Agreement Models for Semi-Supervised Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. 8713–8723.

[34] Yizhou Sun and Jiawei Han. 2012. *Mining Heterogeneous Information Networks: Principles and Methodologies*. Morgan & Claypool Publishers.

[35] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 807–816.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations (ICLR)* (2018). `https://openreview.net/forum?id=rJXMpikCZ`

[37] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning (ICML)*. 6861–6871.

[38] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, Vol. 80. PMLR, 5449–5458.

[39] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning (ICML)*. PMLR, 40–48.

[40] J.S. Yedidia, W.T. Freeman, and Y. Weiss. 2003. Understanding Belief Propagation and its Generalizations. *Exploring Artificial Intelligence in the New Millennium* 8 (2003), 236–239.

[41] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. 2019. Graph convolutional networks: a comprehensive review. *Computational Social Networks* (2019).

[42] Z. Zhang, P. Cui, and W. Zhu. 2020. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* (2020).

[43] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. 2020. Graph Neural Networks with Heterophily. *arXiv preprint arXiv:2009.13566* (2020).

[44] Xiaojin Zhu. 2005. *Semi-supervised learning with graphs*. Ph.D. Dissertation. Carnegie Mellon University, Pittsburgh, PA, USA. `http://portal.acm.org/citation.cfm?id=1104523`

A Nomenclature

We summarize the main symbols used in this work and their definitions below:

Table A.1: Major symbols and definitions.

Symbols	**Definitions**
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	graph $\mathcal{G}$ with nodeset $\mathcal{V}$, edgeset $\mathcal{E}$
$\mathbf{A}$	$n \times n$ adjacency matrix of $\mathcal{G}$
$\mathbf{X}$	$n \times F$ node feature matrix of $\mathcal{G}$
$\mathbf{x}_v$	F-dimensional feature vector for node v
$\mathbf{L}$	unnormalized graph Laplacian matrix
$\mathcal{Y}$	set of class labels
y_v	class label for node $v \in \mathcal{V}$
$\mathbf{y}$	n-dimensional vector of class labels (for all the nodes)
$\mathcal{T}_\mathcal{V} = \{(v_1, y_1), (v_2, y_2), ...\}$	training data for semi-supervised node classification
$N(v)$	general type of neighbors of node v in graph $\mathcal{G}$
$\bar{N}(v)$	general type of neighbors of node v in $\mathcal{G}$ *without self-loops* (i.e., excluding v)
$N_i(v), \bar{N}_i(v)$	i-hop/step neighbors of node v in $\mathcal{G}$ (at exactly distance i) maybe-with/without self-loops, resp.
$\mathcal{E}_2$	set of pairs of nodes (u, v) with shortest distance between them being 2
$d, d_{\max}$	node degree, and maximum node degree across all nodes $v \in \mathcal{V}$, resp.
h	edge homophily ratio
$\mathbf{H}$	class compatibility matrix
$\mathbf{r}_v^{(k)}$	node representations learned in GNN model at round / layer k
K	the number of rounds in the neighborhood aggregation stage
$\mathbf{W}$	learnable weight matrix for GNN model
σ	non-linear activation function
$\|$	vector concatenation operator
`AGGR`	function that aggregates node feature representations within a neighborhood
`COMBINE`	function that combines feature representations from different neighborhoods

B Homophily and Heterophily: Compatibility Matrix

As we mentioned in § 2, the edge homophily ratio in Definition 1 gives an *overall trend* for all the edges in the graph. The actual level of homophily may vary within different pairs of node classes, i.e., there is different tendency of connection between each pair of classes. For instance, in an online purchasing network [24] with three classes—fraudsters, accomplices, and honest users—, fraudsters connect with higher probability to accomplices and honest users. Moreover, within the same network, it is possible that some pairs of classes exhibit homophily, while others exhibit heterophily. In belief propagation [40], a message-passing algorithm used for inference on graphical models, the different levels of homophily or affinity between classes are captured via the *class compatibility*, *propagation* or *coupling* matrix, which is typically pre-defined based on domain knowledge. In this work, we define the empirical *class compatibility matrix* $\mathbf{H}$ as follows:

Definition 4 *The class compatibility matrix $\mathbf{H}$ has entries $[\mathbf{H}]_{i,j}$ that capture the fraction of outgoing edges from a node in class i to a node in class j:*

$$[\mathbf{H}]_{i,j} = \frac{|\{(u,v) : (u,v) \in \mathcal{E} \wedge y_u = i \wedge y_v = j\}|}{|\{(u,v) : (u,v) \in \mathcal{E} \wedge y_u = i\}|}$$

By definition, the class compatibility matrix is a stochastic matrix, with each row summing up to 1.

C Proofs and Discussions of Theorems

C.1 Detailed Analysis of Theorem 1

Proof 1 (for Theorem 1) *We first discuss the GCN layer formulated as $f(\mathbf{X}; \mathbf{A}, \mathbf{W}) = (\mathbf{A}+\mathbf{I})\mathbf{X}\mathbf{W}$. Given training set $\mathcal{T}_\mathcal{V}$, the goal of the training process is to optimize the weight matrix $\mathbf{W}$ to minimize the loss function $\mathcal{L}([(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_\mathcal{V},:}\mathbf{W}, [\mathbf{Y}]_{\mathcal{T}_\mathcal{V},:})$, where $[\mathbf{Y}]_{\mathcal{T}_\mathcal{V},:}$ is the one-hot encoding of class labels provided in the training set, and $[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_\mathcal{V},:}\mathbf{W}$ is the predicted probability distribution of class labels for each node v in the training set $\mathcal{T}_\mathcal{V}$.*

Without loss of generality, we reorder $\mathcal{T}_\mathcal{V}$ accordingly such that the one-hot encoding of labels for nodes in training set $[\mathbf{Y}]_{\mathcal{T}_\mathcal{V},:}$ is in increasing order of the class label y_v:

$$[\mathbf{Y}]_{\mathcal{T}_\mathcal{V},:} = \left[\begin{array}{ccccc}
1 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & 0 & 0 & \cdots & 0 \\
\hdashline
0 & 1 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 1 & 0 & \cdots & 0 \\
\hdashline
\vdots & \vdots & \vdots & \ddots & \vdots \\
\hdashline
0 & 0 & 0 & \cdots & 1 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{array}\right]_{|\mathcal{V}|\times|\mathcal{Y}|} \tag{9}$$

Now we look into the term $[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_\mathcal{V},:}$, which is the aggregated feature vectors within neighborhood N_1 for nodes in the training set. Since we assumed that all nodes in $\mathcal{T}_\mathcal{V}$ have degree d, proportion h of their neighbors belong to the same class, while proportion $\frac{1-h}{|\mathcal{Y}|-1}$ of them belong to any other class uniformly, and one-hot representations of node features $\mathbf{x}_v = \text{onehot}(y_v)$ for each node v, we obtain:

$$[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_\mathcal{V},:} = \left[\begin{array}{ccccc}
hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\
\hdashline
\frac{1-h}{|\mathcal{Y}|-1}d & hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{1-h}{|\mathcal{Y}|-1}d & hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\
\hdashline
\vdots & \vdots & \vdots & \ddots & \vdots \\
\hdashline
\frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & hd+1 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & hd+1
\end{array}\right]_{|\mathcal{V}|\times|\mathcal{Y}|} \tag{10}$$

For $[\mathbf{Y}]_{\mathcal{T}_\mathcal{V},:}$ and $[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_\mathcal{V},:}$ that we derived in Eq. (9) and (10), we can find an optimal weight matrix $\mathbf{W}_*$ such that $[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_\mathcal{V},:}\mathbf{W}_* = [\mathbf{Y}]_{\mathcal{T}_\mathcal{V},:}$, making the loss $\mathcal{L}([(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_\mathcal{V},:}\mathbf{W}_*, [\mathbf{Y}]_{\mathcal{T}_\mathcal{V},:}) = 0$. We can use the following way to find $\mathbf{W}_*$: First, sample one node from each class to form a smaller *set $\mathcal{T}_S \subset \mathcal{T}_{\mathcal{V}}$, therefore we have:*

$$[\mathbf{Y}]_{\mathcal{T}_S,:} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}_{|\mathcal{Y}|\times|\mathcal{Y}|} = \mathbf{I}_{|\mathcal{Y}|\times|\mathcal{Y}|}$$

and

$$[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_S,:} = \begin{bmatrix} hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\ \frac{1-h}{|\mathcal{Y}|-1}d & hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & hd+1 \end{bmatrix}_{|\mathcal{Y}|\times|\mathcal{Y}|}$$

Note that $[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_S,:}$ is a circulant matrix, therefore its inverse exists. Using the Sherman-Morrison formula, we can find its inverse as:

$$([(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_S,:})^{-1} = \frac{1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)} \cdot \begin{bmatrix} (|\mathcal{Y}|-1)+(|\mathcal{Y}|-2+h)d & (h-1)d & \cdots & (h-1)d \\ (h-1)d & (|\mathcal{Y}|-1)+(|\mathcal{Y}|-2+h)d & \cdots & (h-1)d \\ \vdots & \vdots & \ddots & \vdots \\ (h-1)d & (h-1)d & \cdots & (|\mathcal{Y}|-1)+(|\mathcal{Y}|-2+h)d \end{bmatrix}$$

Let $\mathbf{W}_* = ([(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_S,:})^{-1}$, and we have $[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_S,:}\mathbf{W}_* = [\mathbf{Y}]_{\mathcal{T}_S,:} = \mathbf{I}_{|\mathcal{Y}|\times|\mathcal{Y}|}$. It is also easy to verify that $[(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_{\mathcal{V}},:}\mathbf{W}_* = [\mathbf{Y}]_{\mathcal{T}_{\mathcal{V}},:}$. $\mathbf{W}_* = ([(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_S,:})^{-1}$ is the optimal weight matrix we can learn under $\mathcal{T}_{\mathcal{V}}$, since it satisfies $\mathcal{L}([(\mathbf{A}+\mathbf{I})\mathbf{X}]_{\mathcal{T}_{\mathcal{V}},:}\mathbf{W}_*, [\mathbf{Y}]_{\mathcal{T}_{\mathcal{V}},:}) = 0$.

Now consider an arbitrary training datapoint $(v, y_v) \in \mathcal{T}_{\mathcal{V}}$, and a perturbation added to the neighborhood $N(v)$ of node v, such that the number of nodes with a randomly selected class label $y_p \in \mathcal{Y} \neq y_v$ is δ_1 less than expected in $N(v)$. We denote the perturbed graph adjacency matrix as $\mathbf{A}_\Delta$. Without loss of generality, we assume node v has $y_v = 1$, and the perturbed class is $y_p = 2$. In this case we have

$$[(\mathbf{A}_\Delta+\mathbf{I})\mathbf{X}]_{v,:} = \begin{bmatrix} hd+1 & \frac{1-h}{|\mathcal{Y}|-1}d - \delta_1 & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \end{bmatrix}$$

Applying the optimal weight matrix we learned on $\mathcal{T}_{\mathcal{V}}$ to the aggregated feature on the perturbed neighborhood $[(\mathbf{A}_\Delta+\mathbf{I})\mathbf{X}]_{v,:}$, we obtain $[(\mathbf{A}_\Delta+\mathbf{I})\mathbf{X}]_{v,:}\mathbf{W}_*$ which equals to:

$$\begin{bmatrix} 1 - \frac{(h-1)d\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)} & -\frac{((|\mathcal{Y}|-1)+(|\mathcal{Y}|-2+h)d)\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)} & -\frac{(h-1)d\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)} & \cdots & -\frac{(h-1)d\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)} \end{bmatrix}$$

Notice that we always have $1 - \frac{(h-1)d\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)} > -\frac{(h-1)d\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)}$, thus the GCN layer formulated as $(\mathbf{A}+\mathbf{I})\mathbf{X}\mathbf{W}$ would misclassify only if the following inequality holds:

$$1 - \frac{(h-1)d\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)} < -\frac{((|\mathcal{Y}|-1)+(|\mathcal{Y}|-2+h)d)\delta_1}{(d+1)(|\mathcal{Y}|-1+(|\mathcal{Y}|h-1)d)}$$

Solving the above inequality for δ_1, we get the amount of perturbation needed as

$$\begin{cases} \delta_1 > \frac{-h|\mathcal{Y}|d-|\mathcal{Y}|+d+1}{|\mathcal{Y}|-1}, \text{when } 0 \le h < \frac{-|\mathcal{Y}|+d+1}{|\mathcal{Y}|d} \\ \delta_1 < \frac{-h|\mathcal{Y}|d-|\mathcal{Y}|+d+1}{|\mathcal{Y}|-1}, \text{when } h > \frac{-|\mathcal{Y}|+d+1}{|\mathcal{Y}|d} \end{cases} \tag{11}$$

and the least absolute amount of perturbation needed is $|\delta_1| = |\frac{-h|\mathcal{Y}|d-|\mathcal{Y}|+d+1}{|\mathcal{Y}|-1}|$.

Now we move on to discuss the GCN layer formulated as $f(\mathbf{X}; \mathbf{A}, \mathbf{W}) = \mathbf{AXW}$ without self loops. Following similar derivations, we obtain the optimal weight matrix $\mathbf{W}_*$ which makes $\mathcal{L}([\mathbf{AX}]_{\mathcal{T}_{\mathcal{V}},:}\mathbf{W}_*, [\mathbf{Y}]_{\mathcal{T}_{\mathcal{V}},:}) = 0$ as:

$$\mathbf{W}_* = ([\mathbf{AX}]_{\mathcal{T}_S,:})^{-1} = \frac{1}{(1-h|\mathcal{Y}|)d} \begin{bmatrix} -(|\mathcal{Y}|-2+h) & 1-h & \cdots & 1-h \\ 1-h & -(|\mathcal{Y}|-2+h) & \cdots & 1-h \\ \vdots & \vdots & \ddots & \vdots \\ 1-h & 1-h & \cdots & -(|\mathcal{Y}|-2+h) \end{bmatrix} \tag{12}$$

Again if for an arbitrary $(v, y_v) \in \mathcal{T_V}$, a perturbation is added to the neighborhood $N(v)$ of the node v, such that the number of nodes with a randomly selected class label $y_p \in \mathcal{Y} \neq y_v$ is δ_2 less than expected in $N(v)$, we have:

$$[\mathbf{A}_\triangle \mathbf{X}]_{v,:} = \left[\begin{array}{ccccc} hd & \frac{1-h}{|\mathcal{Y}|-1}d - \delta_2 & \frac{1-h}{|\mathcal{Y}|-1}d & \cdots & \frac{1-h}{|\mathcal{Y}|-1}d \end{array} \right]$$

Then applying the optimal weight matrix that we learned on $\mathcal{T_V}$ to the aggregated feature on perturbed neighborhood $[\mathbf{A}_\triangle \mathbf{X}]_{v,:}$, we obtain $[\mathbf{A}_\triangle \mathbf{X}]_{v,:}\mathbf{W}_*$ which equals to:

$$\left[\begin{array}{ccccc} 1 - \frac{(1-h)\delta_2}{(1-h|\mathcal{Y}|)d} & \frac{(|\mathcal{Y}|-2+h)\delta_2}{(1-h|\mathcal{Y}|)d} & -\frac{(1-h)\delta_2}{(1-h|\mathcal{Y}|)d} & \cdots & -\frac{(1-h)\delta_2}{(1-h|\mathcal{Y}|)d} \end{array} \right]$$

Thus, the GCN layer formulated as $\mathbf{AXW}$ would misclassify when the following inequality holds:

$$1 - \frac{(1-h)\delta_2}{(1-h|\mathcal{Y}|)d} < \frac{(|\mathcal{Y}|-2+h)\delta_2}{(1-h|\mathcal{Y}|)d}$$

Or the amount of perturbation is:

$$\begin{cases} \delta_2 > \frac{(1-h|\mathcal{Y}|)d}{|\mathcal{Y}|-1}, \textit{when } 0 \le h < \frac{1}{|\mathcal{Y}|} \\ \delta_2 < \frac{(1-h|\mathcal{Y}|)d}{|\mathcal{Y}|-1}, \textit{when } h > \frac{1}{|\mathcal{Y}|} \end{cases} \tag{13}$$

As a result, the least absolute amount of perturbation needed is $|\delta_2| = |\frac{(1-h|\mathcal{Y}|)d}{|\mathcal{Y}|-1}|$.

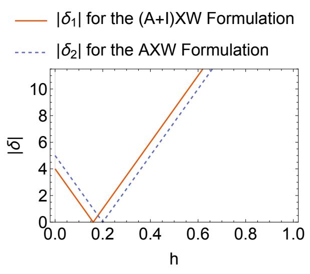
By comparing the least absolute amount of perturbation needed for both formulations to misclassify ($|\delta_1| = |\frac{-h|\mathcal{Y}|d-|\mathcal{Y}|+d+1}{|\mathcal{Y}|-1}|$ derived in Eq. (11) for the $(\mathbf{A}+\mathbf{I})\mathbf{XW}$ formulation; $|\delta_2| = |\frac{(1-h|\mathcal{Y}|)d}{|\mathcal{Y}|-1}|$ derived in Eq. (13) for the $\mathbf{AXW}$ formulation), we can see that $|\delta_1| = |\delta_2|$ if and only if $\delta_1 = -\delta_2$, which happens when $h = \frac{1-|\mathcal{Y}|+2d}{2|\mathcal{Y}|d}$. When $h < \frac{1-|\mathcal{Y}|+2d}{2|\mathcal{Y}|d}$ (heterophily), we have $|\delta_1| < |\delta_2|$, which means the $(\mathbf{A}+\mathbf{I})\mathbf{XW}$ formulation is less robust to perturbation than the $\mathbf{AXW}$ formulation. ■

Discussions From the above proof, we can see that the least absolute amount of perturbation $|\delta|$ needed for both GCN formulations is a function of the assumed homophily ratio h, the node degree d for each node in the training set $\mathcal{T_V}$, and the size of the class label set $|\mathcal{Y}|$. Fig. 4 shows the plots of $|\delta_1|$ and $|\delta_2|$ as functions of h, $|\mathcal{Y}|$ and d: from Fig. 4a, we can see that the least absolute amount of perturbations $|\delta|$ needed for both formulation first decreases as the assumed homophily level h increases, until δ reaches 0, where the GCN layer predicts the same probability for all class labels; after that, δ decreases further below 0, and $|\delta|$ increases as h increases; the $(\mathbf{A}+\mathbf{I})\mathbf{XW}$ formulation is less robust to perturbation than the $\mathbf{AXW}$ formulation at low homophily level until $h = \frac{1-|\mathcal{Y}|+2d}{2|\mathcal{Y}|d}$ as our proof shows, where $|\delta_1| = |\delta_2|$. Figure 4b shows the changes of $|\delta|$ as a function of $|\mathcal{Y}|$ when fixed $h = 0.1$ and $d = 20$. For both formulations, $|\delta|$ first decrease rapidly as $|\mathcal{Y}|$ increases until δ reaches 0, after that δ increases slowly as $|\mathcal{Y}|$ increases; this reveals that both GCN formulations are more robust when $|\mathcal{Y}| << d$ under high homophily level, and in that case $\mathbf{AXW}$ formulation is more robust than the $(\mathbf{A}+\mathbf{I})\mathbf{XW}$ formulation. Figure 4c shows the changes of $|\delta|$ as a function of d for fixed $h = 0.1$ and $|\mathcal{Y}| = 5$: in this case the $\mathbf{AXW}$ formulation is always more robust than the $(\mathbf{A}+\mathbf{I})\mathbf{XW}$ formulation, and for the $(\mathbf{A}+\mathbf{I})\mathbf{XW}$ formulation, $|\delta|$ follows again a “V”-shape curve as d changes.

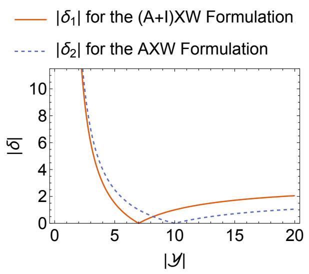
C.2 Detailed Analysis of Theorem 2

Proof 2 (for Theorem 2) *For all $v \in \mathcal{V}$, since its neighbors' class labels $\{y_u : u \in N(v)\}$ are conditionally independent given y_v, we can define a matrix $\mathbf{P}_v$ for each node v as $[\mathbf{P}_v]_{i,j} = P(y_u = j|y_v = i), \forall i, j \in \mathcal{Y}, u \in N(v)$. Following the assumption that for all $v \in \mathcal{V}$, $P(y_u = y_v|y_v) = h$, $P(y_u = y|y_v) = \frac{1-h}{|\mathcal{Y}|-1}, \forall y \neq y_v$, we have*

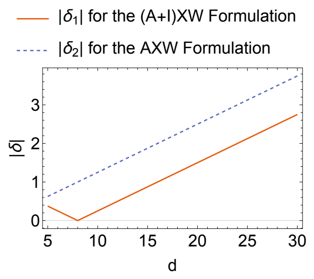
$$\mathbf{P}_v = \mathbf{P} = \left[\begin{array}{cccc} h & \frac{1-h}{|\mathcal{Y}|-1} & \cdots & \frac{1-h}{|\mathcal{Y}|-1} \\ \frac{1-h}{|\mathcal{Y}|-1} & h & \cdots & \frac{1-h}{|\mathcal{Y}|-1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1-h}{|\mathcal{Y}|-1} & \frac{1-h}{|\mathcal{Y}|-1} & \cdots & h \end{array} \right], \forall v \in \mathcal{V} \tag{14}$$



(a) $|\delta|$ as a function of h under $d = 20, |\mathcal{Y}| = 5$.



(b) $|\delta|$ as a function of $|\mathcal{Y}|$ under $h = 0.1, d = 20$.



(c) $|\delta|$ as a function of d under $h = 0.1, |\mathcal{Y}| = 5$.

Figure 4: Perturbation $|\delta|$ needed in order for GCN layers $(\mathbf{A} + \mathbf{I})\mathbf{XW}$ and $\mathbf{AXW}$ to misclassify a node: Examples of perturbation $|\delta|$ as functions of h, $|\mathcal{Y}|$ and d, respectively.

Now consider node $w \in N_2(v)$, we have:

$$P(y_w = k|y_v = i) = \sum_{j \in |\mathcal{Y}|} P(y_w = k|y_u = j)P(y_u = j|y_v = i) = \sum_{j \in |\mathcal{Y}|} [\mathbf{P}]_{j,k}[\mathbf{P}]_{i,j} = \mathbf{P}^2 \quad (15)$$

Therefore, to prove that the 2-hop neighborhood $N_2(v)$ for any node $v \in \mathcal{V}$ is homophily-dominant in expectation (i.e. $P(y_w = i|y_v = i) \geq P(y_w = j|y_v = i), \forall j \in \mathcal{Y} \neq i, w \in N_2(v)$), we need to show that the diagonal entries $[\mathbf{P}^2]_{i,i}$ of $\mathbf{P}^2$ are larger than the off-diagonal entries $[\mathbf{P}^2]_{i,j}$.

Denote $\rho = \frac{1-h}{|\mathcal{Y}|-1}$. From Eq. (14)*, we have*

$$[\mathbf{P}^2]_{i,i} = h^2 + (|\mathcal{Y}| - 1)\rho^2 \quad (16)$$

and for $i \neq j$

$$[\mathbf{P}^2]_{i,j} = 2h\rho + (|\mathcal{Y}| - 2)\rho^2 \quad (17)$$

Thus,

$$[\mathbf{P}^2]_{i,i} - [\mathbf{P}^2]_{i,j} = h^2 - 2h\rho + \rho^2 = (h - \rho)^2 \geq 0$$

with equality if and only if $h = \rho$, namely $h = \frac{1}{|\mathcal{Y}|}$. Therefore, we proved that the 2-hop neighborhood $N_2(v)$ for any node $v \in \mathcal{V}$ will always be homophily-dominant in expectation. ■

C.3 Detailed Analysis of Theorem 3

Preliminaries We define unnormalized Laplacian matrix of graph $\mathcal{G}$ as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{A} \in \{0, 1\}^{|\mathcal{V}| \times |\mathcal{V}|}$ is the adjacency matrix and $\mathbf{D}$ is the diagonal matrix with $[\mathbf{D}]_{i,i} = \sum_j [\mathbf{A}]_{i,j}$. Without loss of generality, since the eigenvalues $\{\lambda_i\}$ of $\mathbf{L}$ are real and nonnegative [32], we assume the following order for the eigenvalues of $\mathbf{L}$: $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{|\mathcal{V}|-1} = \lambda_{max}$. Furthermore, since $\mathbf{L}$ is real and symmetric, there exists a set of orthonormal eigenvectors $\{\mathbf{v}_i\}$ that form a complete basis of $\mathbb{R}^{|\mathcal{V}|}$. This means that for any graph signal $\mathrm{s} \in \mathbb{R}^{|\mathcal{V}|}$, where s_u is the value of the signal on node $u \in \mathcal{V}$, it can be decomposed to a weighted sum of $\{\mathbf{v}_i\}$. Mathematically, s is represented as $\mathrm{s} = \sum_{i=0}^{|\mathcal{V}|-1} c_{s,i}\mathbf{v}_i$, where $c_{s,i} = \mathrm{s}^\mathsf{T}\mathbf{v}_i$. We regard $c_{s,i}$ as the coefficient of s at frequency component i and regard the coefficients at all frequencies components $\{c_{s,i}\}$ as the spectrum of signal s with respect to graph $\mathcal{G}$. In the above order of the eigenvalues, λ_i which are closer to 0 would correspond to lower-frequency components, and λ_i which are closer to λ_{max} would correspond to higher-frequency components. Interested readers are referred to [32] for further details regarding signal processing on graphs.

The **smoothness score of a signal** s on graph $\mathcal{G}$, which measures the amount of changes of signal s along the edges of graph $\mathcal{G}$, can be defined using $\mathbf{L}$ as

$$\mathrm{s}^\mathsf{T}\mathbf{L}\mathrm{s} = \sum_{i,j} \mathbf{A}_{ij}(\mathrm{s}_i - \mathrm{s}_j)^2 = \sum_{u \in \mathcal{V}} \sum_{v \in N(u)} (\mathrm{s}_u - \mathrm{s}_v)^2. \quad (18)$$

Then, for two eigenvectors $\mathbf{v}_i$ and $\mathbf{v}_j$ corresponding to eigenvalues $\lambda_i \leq \lambda_j$ of $\mathbf{L}$, we have:

$$\mathbf{v}_i^\mathsf{T}\mathbf{L}\mathbf{v}_i = \lambda_i \leq \lambda_j = \mathbf{v}_j^\mathsf{T}\mathbf{L}\mathbf{v}_j$$

which means that $\mathbf{v}_i$ is *more smooth* than $\mathbf{v}_j$. This matches our expectations that a lower-frequency signal on $\mathcal{G}$ should have smaller smoothness score. The **smoothness score for arbitrary graph signal** $\mathbf{s} \in \mathbb{R}^{|\mathcal{V}|}$ can be represented by its coefficients of each frequency component as:

$$\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} = \left(\sum_i c_{s,i}\mathbf{v}_i\right)\mathbf{L}\left(\sum_i c_{s,i}\mathbf{v}_i\right) = \sum_{i=0}^{|\mathcal{V}|-1} c_{s,i}^2\lambda_i \tag{19}$$

with the above preliminaries, we can define the following concept:

Definition 5 *Suppose* $\mathbf{s} = \sum_{i=0}^{|\mathcal{V}|-1} c_{s,i}\mathbf{v}_i$ *and* $\mathbf{t} = \sum_{i=0}^{|\mathcal{V}|-1} c_{t,i}\mathbf{v}_i$ *are two graph signals defined on* $\mathcal{G}$. *In the spectrum of the unnormalized graph laplacian* $\mathbf{L}$, *graph signal* $\mathbf{s}$ *has higher energy on high-frequency components than* $\mathbf{t}$ *if there exists integer* $0 < M \leq |\mathcal{V}| - 1$ *such that* $\sum_{i=M}^{|\mathcal{V}|-1} c_{s,i}^2 > \sum_{i=M}^{|\mathcal{V}|-1} c_{t,i}^2$.

Based on these preliminary definitions, we can now proceed with the proof of the theorem:

Proof 3 (for Theorem 3) *We first prove that for graph signals* $\mathbf{s}, \mathbf{t} \in \{0,1\}^{|\mathcal{V}|}$, *edge homophily ratio* $h_s < h_t$ *if and only if* $\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} > \mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t}$. *Following Dfn. 1, the edge homophily ratio for signal* $\mathbf{s}$ *(similarly for* $\mathbf{t}$*) can be calculated as:*

$$h_s = \frac{1}{2|\mathcal{E}|}\sum_{u\in\mathcal{V}}\left(d_u - \sum_{v\in N(v)}(\mathbf{s}_u - \mathbf{s}_v)^2\right) = \frac{1}{2|\mathcal{E}|}\sum_{u\in\mathcal{V}} d_u - \frac{1}{2|\mathcal{E}|}\sum_{u\in\mathcal{V}}\sum_{v\in N(v)}(\mathbf{s}_u - \mathbf{s}_v)^2 \tag{20}$$

Plugging this in Eq. (18), *we obtain:*

$$h_s = \frac{1}{2|\mathcal{E}|}\sum_{u\in\mathcal{V}} d_u - \frac{1}{2|\mathcal{E}|}\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} = 1 - \frac{1}{2|\mathcal{E}|}\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s}$$

where $|\mathcal{E}|$ *is the number of edges in* $\mathcal{G}$. *From the above equation, we have*

$$h_s < h_t \;\Leftrightarrow\; 1 - \frac{1}{2|\mathcal{E}|}\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} < 1 - \frac{1}{2|\mathcal{E}|}\mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t} \;\Leftrightarrow\; \mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} > \mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t}$$

i.e. edge homophily ratio $h_s < h_t$ *if and only if* $\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} > \mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t}$.

Next we prove that if $\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} > \mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t}$, *then following Dfn.5, signal* $\mathbf{s}$ *has higher energy on high-frequency components than* $\mathbf{t}$. *We prove this by contradiction: suppose integer* $0 < M \leq |\mathcal{V}| - 1$ *does not exist such that* $\sum_{i=M}^{|\mathcal{V}|-1} c_{si}^2 > \sum_{i=M}^{|\mathcal{V}|-1} c_{ti}^2$ *when* $\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} > \mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t}$, *then all of the following inequalities must hold, as the eigenvalues of* $\mathbf{L}$ *satisfy* $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{|\mathcal{V}|-1} = \lambda_{max}$*:*

$$\begin{gathered}
0 = \lambda_0(c_{s,0}^2 + c_{s,1}^2 + c_{s,2}^2 + \cdots + c_{s,|\mathcal{V}|-1}^2) = \lambda_0(c_{t,0}^2 + c_{t,1}^2 + c_{t,2}^2 + \cdots + c_{t,|\mathcal{V}|-1}^2) = 0 \\
(\lambda_1 - \lambda_0)(c_{s,1}^2 + c_{s,2}^2 + \cdots + c_{s,|\mathcal{V}|-1}^2) \leq (\lambda_1 - \lambda_0)(c_{t,1}^2 + c_{t,2}^2 + \cdots + c_{t,|\mathcal{V}|-1}^2) \\
(\lambda_2 - \lambda_1)(c_{s,2}^2 + \cdots + c_{s,|\mathcal{V}|-1}^2) \leq (\lambda_2 - \lambda_1)(c_{t,2}^2 + \cdots + c_{t,|\mathcal{V}|-1}^2) \\
\vdots \\
(\lambda_{|\mathcal{V}|-1} - \lambda_{|\mathcal{V}|-2})c_{s,|\mathcal{V}|-1}^2 \leq (\lambda_{|\mathcal{V}|-1} - \lambda_{|\mathcal{V}|-2})c_{t,|\mathcal{V}|-1}^2
\end{gathered}$$

Summing over both sides of all the above inequalities, we have

$$\lambda_0\cdot c_{s,0}^2 + \lambda_1\cdot c_{s,1}^2 + \lambda_2\cdot c_{s,2}^2 + \cdots + \lambda_{|\mathcal{V}|-1}\cdot c_{s,|\mathcal{V}|-1}^2 \leq \lambda_0\cdot c_{t,0}^2 + \lambda_1\cdot c_{t,1}^2 + \lambda_2\cdot c_{t,2}^2 + \cdots + \lambda_{|\mathcal{V}|-1}\cdot c_{t,|\mathcal{V}|-1}^2$$

i.e., $\sum_{i=0}^{|\mathcal{V}|-1} c_{si}^2\lambda_i \leq \sum_{i=0}^{|\mathcal{V}|-1} c_{ti}^2\lambda_i$. *However, from Eq.* (19), *we should have*

$$\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} > \mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t} \;\Leftrightarrow\; \sum_{i=0}^{|\mathcal{V}|-1} c_{si}^2\lambda_i > \sum_{i=0}^{|\mathcal{V}|-1} c_{ti}^2\lambda_i$$

which contradicts with the previous resulting inequality. Therefore, the assumption should not hold, and there must exist an integer $0 < M \leq |\mathcal{V}| - 1$ *such that* $\sum_{i=M}^{|\mathcal{V}|-1} c_{si}^2 > \sum_{i=M}^{|\mathcal{V}|-1} c_{ti}^2$ *when* $\mathbf{s}^\mathsf{T}\mathbf{L}\mathbf{s} > \mathbf{t}^\mathsf{T}\mathbf{L}\mathbf{t}$. ■

Extension of Theorem 3 to one-hot encoding of class label vectors Theorem 3 discusses only the graph signal $\mathbf{s}, \mathbf{t} \in \{0,1\}^{|\mathcal{V}|}$ with only 1 channel (i.e., with only 1 value assigned to each node). It is possible to generalize the theorem to one-hot encoding $\mathbf{Y}_s, \mathbf{Y}_t \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{Y}|}$ as graph signal with $|\mathcal{Y}|$-channels by modifying Dfn. 5 as follows:

Definition 6 *Suppose* $[\mathbf{Y}_s]_{:,j} = \sum_{i=0}^{|\mathcal{V}|-1} c_{s,j,i}\mathbf{v}_i$ *and* $[\mathbf{Y}_t]_{:,j} = \sum_{i=0}^{|\mathcal{V}|-1} c_{t,j,i}\mathbf{v}_i$ *are one-hot encoding of class label vector* $\mathbf{y}_s, \mathbf{y}_t$ *defined as graph signals on* $\mathcal{G}$*, where* $c_{s,j,i} = [\mathbf{Y}_s]_{:,j}^{\mathsf{T}}\mathbf{v}_i$ *is the coefficient of the* j*th-channel of* $\mathbf{Y}_s$ *at frequency component* i*. In the spectrum of the unnormalized graph laplacian* $\mathbf{L}$*, graph signal* $\mathbf{Y}_s$ *has higher energy on high-frequency components than* $\mathbf{Y}_t$ *if there exists integer* $0 < M \leq |\mathcal{V}| - 1$ *such that* $\sum_{i=M}^{|\mathcal{V}|-1}\sum_{j=1}^{\phi} c_{s,j,i}^2 > \sum_{i=M}^{|\mathcal{V}|-1}\sum_{j=1}^{\phi} c_{t,j,i}^2$*.*

Under this definition, we can prove Theorem 3 for one-hot encoding of class label vectors $\mathbf{Y}_s, \mathbf{Y}_t$ as before, with the modification that in this case we have for signal $\mathbf{Y}_s$ (similarly for $\mathbf{Y}_t$):

$$h_s = \frac{1}{4|\mathcal{E}|}\sum_{u\in\mathcal{V}}\left(2d_u - \sum_{v\in N(v)}\sum_{j=1}^{\phi}([\mathbf{Y}_s]_{u,j} - [\mathbf{Y}_t]_{v,j})^2\right)$$

instead of Eq. (20). The rest of the proof is similar to Proof 3.

D Our H$_2$GCN model: Details

In this section, we give the pipeline and pseudocode of H$_2$GCN, elaborate on its differences from existing GNN models, and present a detailed analysis of its computational complexity.

D.1 Pseudocode & Pipeline

In Fig. 5 we visualize H$_2$GCN, which we describe in § 3.2. We also give its pseudocode in Algorithm 1.

D.2 Detailed Comparison of H$_2$GCN to existing GNN models

In § 4, we discussed several high-level differences between H$_2$GCN and the various GNN models that we consider in this work, including the inclusion or not of designs D1-D3. Here we give some additional conceptual and mechanism differences.

As we have mentioned, H$_2$GCN differs from GCN [17] in a number of ways: (1) In each round of propagation/aggregation, GCN "mixes" the ego- and neighbor-representations by repeatedly averaging them to obtain the new node representations, while H$_2$GCN keeps them distinct via concatenation; (2) GCN considers only the 1-hop neighbors (including the ego / self-loops), while H$_2$GCN considers higher-order neighborhoods ($\bar{N}_1$ and $\bar{N}_2$); (3) GCN applies non-linear embedding transformations per round (e.g., RELU), while H$_2$GCN perform feature embedding for the ego in the first layer and drops all other non-linearities in the aggregation stage; and (4) GCN does not use the jumping knowledge framework (unlike H$_2$GCN), and makes the node classification predictions based on the last-round representations.

Unlike GAT, H$_2$GCN does not use any attention mechanism. Creating attention mechanisms that can generalize well to heterophily is an interesting future direction. Moreover, GCN-Cheby uses entirely different mechanisms than the other GNN models that we consider (i.e., Chebysev polynomials), though it has some conceptual similarities to H$_2$GCN in terms of the higher-order neighborhoods that it models.

GraphSAGE differs from H$_2$GCN in the same ways that are described in (2)-(4) above. In addition to leveraging only the 1-hop neighborhood, GraphSAGE also samples a fixed number of neighbors per round, while H$_2$GCN uses the full neighborhood. With respect to ego- and neighbor-representations, GraphSAGE concatenates them (as we do) but subsequently applies non-linear embedding transformations to them jointly (while we simplify all non-linear transformations). Our empirical analysis has revealed that such transformations lead to a decrease in performance in heterophily settings (see paragraph below on "Non-linear embedding transformations...").

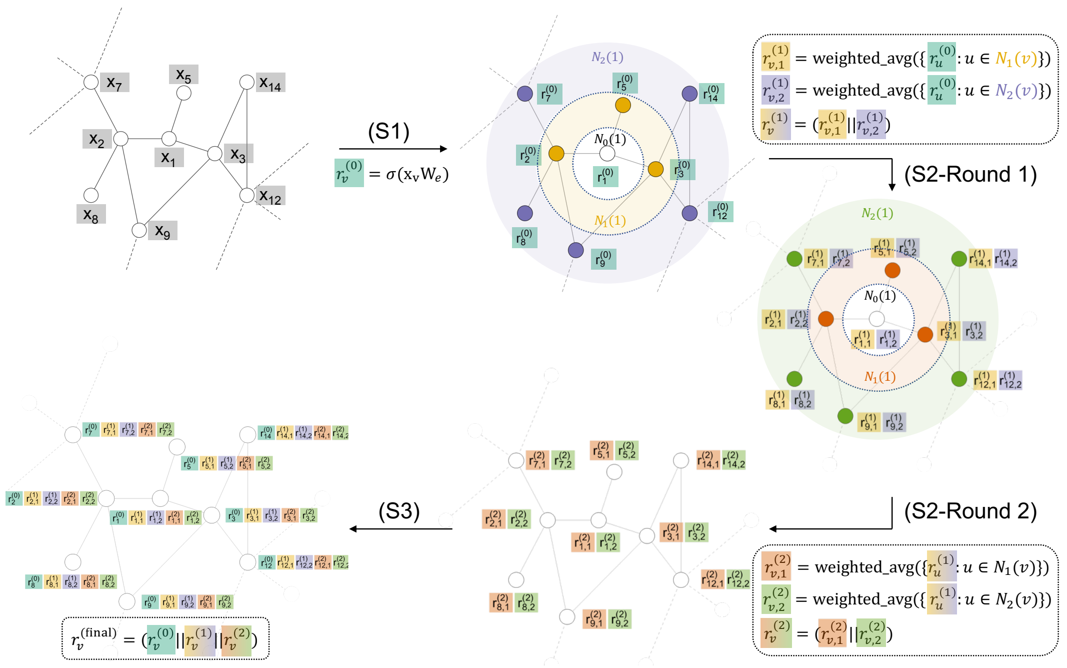

(S1)
$r_v^{(0)} = \sigma(x_v W_e)$
(S2-Round 1)
$r_{v,1}^{(1)}$ = weighted_avg({$r_u^{(0)}$: $u \in N_1(v)$})
$r_{v,2}^{(1)}$ = weighted_avg({$r_u^{(0)}$: $u \in N_2(v)$})
$r_v^{(1)} = (r_{v,1}^{(1)} || r_{v,2}^{(1)})$
(S2-Round 2)
$r_{v,1}^{(2)}$ = weighted_avg({$r_u^{(1)}$: $u \in N_1(v)$})
$r_{v,2}^{(2)}$ = weighted_avg({$r_u^{(1)}$: $u \in N_2(v)$})
$r_v^{(2)} = (r_{v,1}^{(2)} || r_{v,2}^{(2)})$
(S3)
$r_v^{(\text{final})} = (r_v^{(0)} || r_v^{(1)} || r_v^{(2)})$
$N_0(1)$
$N_1(1)$
$N_2(1)$

Figure 5: H$_2$GCN-2 pipeline. It consists of 3 stages: **(S1)** feature embedding, **(S2)** neighborhood aggregation, and **(S3)** classification. The *feature embedding stage* **(S1)** uses a graph-agnostic dense layer to generate the feature embedding $\mathbf{r}_v^{(0)}$ of each node v based on its ego-feature $\mathbf{x}_v$. In the *neighborhood aggregation stage* **(S2)**, the generated embeddings are aggregated and repeatedly updated within the node's neighborhood; the 1-hop neighbors $N_1(v)$ and 2-hop neighbors $N_2(v)$ are aggregated separately and then concatenated, following our design D2. In the *classification stage* **(S3)**, each node is classified based on its final embedding $\mathbf{r}_v^{(\text{final})}$, which consists of its intermediate representations concatenated as per design D3.

Finally, MixHop differs from H$_2$GCN in the same ways that are described in (1) and (3)-(4) above. It explicitly considers higher-order neighborhoods up to N_2, though [1] defines the 2-hop neighborhoods as that including neighbors *up to* 2-hop away neighbors. In our framework, we define the i-hop neighborhood as the set of neighbors with minimum distance exactly i from the ego (§ 2). Finally, the output layer of MixHop uses a tailored, column-wise attention layer, which prioritizes specific features, before the softmax layer. In contrast, before the classification layer, H$_2$GCN uses concatenation-based jumping knowledge in order to represent the high-frequency components that are critical in heterophily.

Non-linear embedding transformations per round in H$_2$GCN? GCN [17], GraphSAGE [11] and other GNN models embed the intermediate representations per round of feature propagation and aggregation. However, as we show in the ablation study in App. G.2 (Table G.4, last row "Non-linear"), introducing non-linear transformations per round of the neighborhood aggregation stage **(S2)** of H$_2$GCN-2 (i.e., with $K = 2$) as follows leads to *worse* performance than the framework design that we introduce in Eq. (5) of § 3.2:

$$\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\sigma\left(\mathbf{W}\left[\mathbf{r}_v^{(k-1)}, \texttt{AGGR}\{\mathbf{r}_u^{(k-1)} : u \in N_1(v)\}, \texttt{AGGR}\{\mathbf{r}_u^{(k-1)} : u \in N_2(v)\}\right]\right)\right), \tag{21}$$

where σ is RELU and $\mathbf{W}$ is a learnable matrix. Our design in Eq. 5 aggregates different neighborhoods in a similar way to SGC [37], which has shown that removing non-linearities does not negatively impact performance in homophily settings. We actually find that removing non-linearities even *improves* the performance under heterophily.

D.3 H$_2$GCN: Time Complexity in Detail

Preliminaries The worst case time complexity for calculating $\mathbf{A} \cdot \mathbf{B}$ when both $\mathbf{A}$ and $\mathbf{B}$ are sparse matrices is $\mathrm{O}(\mathrm{nnz}(\mathbf{A}) \cdot c_{\mathbf{B}})$, where $\mathrm{nnz}(\mathbf{A})$ is the number of non-zero elements in matrix

Algorithm 1: H_2GCN Framework for Node Classification under Homophily & Heterophily

```
Input: Graph Adjacency Matrix $\mathbf{A} \in \{0,1\}^{n\times n}$; Node Feature Matrix $\mathbf{X} \in \mathbb{R}^{n\times F}$; Set of Labels $\mathcal{Y}$;
       Labeled Nodes $\mathcal{T}_\mathcal{V}$
Hyper-parameters: Dropout Rate; Non-linearity function $\sigma$; Number of Embedding Rounds $K$;
                  Dimension of Feature Embedding $p$;
Network Parameters: $\mathbf{W}_e \in \mathbb{R}^{F\times p}$; $\mathbf{W}_c \in \mathbb{R}^{(2^{K+1}-1)p\times|\mathcal{Y}|}$
Output: Class label vector $\mathbf{y}$
begin
    /* All new variables defined below are initialized as all 0 */
    /* Stage S1: Feature Embedding */
    for $v \in \mathcal{V}$ do
        $\mathbf{r}_v^{(0)} \leftarrow \sigma(\mathbf{x}_v \mathbf{W}_e)$ /* Embeddings stored in matrix $\mathbf{R}$ */
    /* Stage S2: Neighborhood Aggregation */
    /* Calculate higher-order neighborhoods $\bar{N}_1$ and $\bar{N}_2$ without self-loops and
       their corresponding adjacency matrices $\bar{\mathbf{A}}_1$ and $\bar{\mathbf{A}}_2$ */
    $\mathbf{A}_0 \leftarrow \mathbf{I}_n$                          /* $\mathbf{I}_n$ is the $n \times n$ identity matrix */
    $\bar{\mathbf{A}}_1 \leftarrow \mathbb{I}[\mathbf{A} - \mathbf{I}_n > 0]$      /* $\mathbb{I}$ is a element-wise indicator function for matrix */
    $\bar{\mathbf{A}}_2 \leftarrow \mathbb{I}\left[\mathbf{A}^2 - \mathbf{A} - \mathbf{I}_n > 0\right]$;
    for $i \leftarrow 1$ to 2 do
        for $v \in \mathcal{V}$ do
            $d_{v,i} \leftarrow \sum_k \bar{a}_{vk,i}$                 /* degree of node $v$ at neighborhood $\bar{N}_i$ */
        $\bar{\mathbf{D}}_i \leftarrow \text{diag}\{d_{v,i} : v \in \mathcal{V}\}$;
        $\bar{\mathbf{A}}_i \leftarrow \bar{\mathbf{D}}_i^{-\frac{1}{2}} \bar{\mathbf{A}}_i \bar{\mathbf{D}}_i^{-\frac{1}{2}}$        /* symmetric degree-normalization of matrices $\bar{\mathbf{A}}_i$ */
    for $k \leftarrow 1$ to $K$ do
        $\mathbf{R}_1^{(k)} \leftarrow \bar{\mathbf{A}}_1 \mathbf{R}^{(k-1)}$                         /* Designs D1 + D2 */
        $\mathbf{R}_2^{(k)} \leftarrow \bar{\mathbf{A}}_2 \mathbf{R}^{(k-1)}$;
        /* $\|$ is the vector concatenation operator */
        $\mathbf{R}^{(k)} \leftarrow \left(\mathbf{R}_1^{(k)} \| \mathbf{R}_2^{(k)}\right)$
    $\mathbf{R}^{(\text{final})} \leftarrow \left(\mathbf{R}^{(0)} \| \mathbf{R}^{(1)} \| \ldots \| \mathbf{R}^{(K)}\right)$            /* Design D3 */
    /* Stage S3: Classification */
    $\mathbf{R}^{(\text{final})} \leftarrow \text{dropout}(\mathbf{R}^{(\text{final})})$              /* default dropout rate: 0.5 */
    for $v \in \mathcal{V}$ do
        $\mathbf{p}_v \leftarrow \text{softmax}(\mathbf{r}_v^{(\text{final})} \mathbf{W}_c)$;
        $\mathbf{y}_v \leftarrow \arg\max(\mathbf{p}_v)$                          /* class label */
```

$\mathbf{A}$, and $c_{\mathbf{B}} = \max(\sum_j \mathbb{I}[b_{ij} > 0])$ is the maximum number of non-zero elements in any row of matrix $\mathbf{B}$. The time complexity for calculating $\mathbf{A} \cdot \mathbf{X}$, when $\mathbf{X}$ is a dense matrix with F columns, is $\text{O}(\text{nnz}(\mathbf{A})F)$.

Time complexity of H_2GCN We analyze the time complexity of H_2GCN by stage (except the classification stage).

The feature embedding stage **(S1)** takes $\text{O}(\text{nnz}(\mathbf{X})p)$ to calculate $\sigma(\mathbf{X}\mathbf{W}_e)$ where $\mathbf{W}_e \in \mathbb{R}^{F\times p}$ is a learnable dense weight matrix, and $\mathbf{X} \in \mathbb{R}^{n\times F}$ is the node feature matrix.

In the neighborhood aggregation stage **(S2)**, we perform the following computations:

- *Calculation of higher-order neighborhoods.* Given that $\mathbf{A}$ is sparse, we can obtain the 2-hop neighborhood by calculating $\mathbf{A}^2$ in $\text{O}(|\mathcal{E}|d_{\max})$, where $|\mathcal{E}|$ is the number of edges in $\mathcal{G}$ (equal to the number of non-zeroes in $\mathbf{A}$), and $d_{\max}$ is the maximum degree across all nodes $v \in \mathcal{V}$ (which is equal to the maximum number of non-zeroes in any row of $\mathbf{A}$).
- *Feature Aggregation.* We begin with a p-dimensional embedding for each node after feature embedding. In round k, since we are using the neighborhoods $\bar{N}_1$ and $\bar{N}_2$, we have an em-

bedding $\mathbf{R}^{(k-1)} \in \mathbb{R}^{n \times 2^{(k-1)}p}$ as input. We aggregate embedding vectors within neighborhood by $\mathbf{R}^{(k)} = \left(\bar{\mathbf{A}}_1 \mathbf{R}^{(k-1)} \| \bar{\mathbf{A}}_2 \mathbf{R}^{(k-1)}\right)$, in which $\bar{\mathbf{A}}_i$ corresponds to the adjacency matrix of neighborhood $\bar{N}_i$. The two sparse matrix-matrix multiplications in the concatenation take $\mathrm{O}\left(|\mathcal{E}|2^{(k-1)}p + |\mathcal{E}_2|2^{(k-1)}p\right)$, where $|\mathcal{E}_2| = \frac{1}{2}\sum_{v \in \mathcal{V}} |\bar{N}_2(v)|$. Over K rounds of embedding, the complexity becomes $\mathrm{O}\left(2^K(|\mathcal{E}| + |\mathcal{E}_2|)p\right)$.

Adding all the big-O terms above, we have the overall time complexity for stages **(S1)** and **(S2)** of H$_2$GCN as:

$$\mathrm{O}\left(\mathrm{nnz}(\mathbf{X})\, p + |\mathcal{E}| d_{\max} + 2^K(|\mathcal{E}| + |\mathcal{E}_2|)p\right),$$

where K is usually a small number (e.g., 2). For small values of K, the complexity becomes $\mathrm{O}\left(|\mathcal{E}| d_{\max} + (\mathrm{nnz}(\mathbf{X}) + |\mathcal{E}| + |\mathcal{E}_2|)p\right)$.

E Additional Related Work

In § 4, we discuss relevant work on GNNs. Here we briefly mention other approaches for node classification.

Collective classification in statistical relational learning focuses on the problem of node classification by leveraging the correlations between the node labels and their attributes [30]. Since exact inference is NP-hard, approximate inference algorithms (e.g., iterative classification [14, 20], loopy belief propagation) are used to solve the problem. Belief propagation (BP) [40] is a classic message-passing algorithm for graph-based semi-supervised learning, which can be used for graphs exhibiting homophily or heterophily [19] and has fast linearized versions [10, 8]. Different from the setup where GNNs are employed, BP does *not* by itself leverage node features, and usually assumes a pre-defined class compatibility or edge potential matrix (§ 2). We note, however, that Gatterbauer [9] proposed estimating the class compatibility matrix instead of using a pre-defined one in the BP formulation. Moreover, the recent CPGNN model [43] integrates the compatibility matrix as a set of learnable parameters into GNN, which it initializes with an estimated class compatibility matrix. Another classic approach for collective classification or graph-based semi-supervised learning is label propagation, which iteratively propagates the (up-to-date) label information of each node to its neighbors in order to minimize the overall smoothness penalty of label assignments in the graph. Standard label propagation approaches inherently assume homophily by penalizing different label assignments among immediate neighborhoods, but more recent works have also looked into formulations which can better address heterophily: Before applying label propagation, Peel [25] transforms the original graph into either a similarity graph by measuring similarity between node neighborhoods or a new graph connecting nodes that are two hops away; Chin et al. [6] decouple graph smoothing where the notion of "identity" and "preference" for each node are considered separately. However, like BP, these approaches do not by themselves utilize node features.

F Experimental Setup & Hyperparameter Tuning

F.1 Setup

H$_2$GCN Implementation We use $K = 1$ for H$_2$GCN-1 and $K = 2$ for H$_2$GCN-2. For loss function, we calculate the cross entropy between the predicted and the ground-truth labels for nodes within the training set, and add L_2 regularization of network parameters $\mathbf{W}_e$ and $\mathbf{W}_c$. (cf. Alg. 1)

Baseline Implementations For all baselines besides MLP, we used the official implementation released by the authors on GitHub.

- **GCN & GCN-Cheby** [17]: `https://github.com/tkipf/gcn`
- **GraphSAGE** [11]: `https://github.com/williamleif/graphsage-simple` (PyTorch implementation)
- **MixHop** [1]: `https://github.com/samihaija/mixhop`
- **GAT** [36]: `https://github.com/PetarV-/GAT`. (For large datasets, we make use of the sparse version provided by the author.)

For MLP, we used our own implementation of MLP with 1-hidden layer, which is equivalent to the case of $K = 0$ in Algorithm 1. We use the same loss function as H_2GCN for training MLP.

Hardware Specifications We run experiments on synthetic benchmarks with an Amazon EC2 instance with instance size as `p3.2xlarge`, which features an 8-core CPU, 61 GB Memory, and a Tesla V100 GPU with 16 GB GPU Memory. For experiments on real benchmarks, we use a workstation with a 12-core AMD Ryzen 9 3900X CPU, 64GB RAM, and a Quadro P6000 GPU with 24 GB GPU Memory.

F.2 Tuning the GNN Models

To avoid bias, we tuned the hyperparameters of each method (H_2GCN and baseline models) on each benchmark. Below we list the hyperparameters tested on each benchmark per model. As the hyperparameters defined by each baseline model differ significantly, we list the combinations of non-default command line arguments we tested, without explaining them in detail. We refer the interested reader to the corresponding original implementations for further details on the arguments, including their definitions.

Synthetic Benchmark Tuning For each synthetic benchmark, we report the results for different heterophily levels under the same set of hyperparameters for each method, so that we can compare how the same hyperparameters perform across the full spectrum of low-to-high homophily. We report the best performance, for the set of hyperparameters which performs the best on the validation set on the majority of the heterophily levels for each method.

For `syn-cora`, we test the following command-line arguments for each baseline method:

- **H_2GCN-1 & H_2GCN-2**:
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: ReLU
  - Dropout Rate: $a \in \{0, 0.5\}$

  We report the best performance, for $a = 0$.
- **GCN** [17]:
  - `hidden1`: $a \in \{16, 32, 64\}$
  - `early_stopping`: $b \in \{40, 100, 200\}$
  - `epochs`: 2000

  We report the best performance, for $a = 32, b = 40$.
- **GCN-Cheby** [17]:
  - Set 1:
    * `hidden1`: $a \in \{16, 32, 64\}$
    * `dropout`: 0.6
    * `weight_decay`: $b \in \{$`1e-5, 5e-4`$\}$
    * `max_degree`: 2
    * `early_stopping`: 40
  - Set 2:
    * `hidden1`: $a \in \{16, 32, 64\}$
    * `dropout`: 0.5
    * `weight_decay`: `5e-4`
    * `max_degree`: 3
    * `early_stopping`: 40

  We report the best performance, for Set 1 with $a = 64, b =$ `5e-4`.
- **GraphSAGE** [11]:
  - `hid_units`: $a \in \{64, 128\}$
  - `lr`: $b \in \{0.1, 0.7\}$
  - `epochs`: 500

We report the performance with $a = 64, b = 0.7$.

- **MixHop** [1]:
  - `hidden_dims_csv`: $a \in \{64, 192\}$
  - `adj_pows`: 0, 1, 2

  We report the performance with $a = 192$.
- **GAT** [36]:
  - `hid_units`: $a \in \{8, 16, 32, 64\}$
  - `n_heads`: $b \in \{1, 4, 8\}$

  We report the performance with $a = 8, b = 8$.
- **MLP**
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: ReLU
  - Dropout Rate: 0.5

For `syn-products`, we test the following command-line arguments for each baseline method:

- **H$_2$GCN-1 & H$_2$GCN-2**:
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: ReLU
  - Dropout Rate: $a \in \{0, 0.5\}$

  We report the best performance, for $a = 0.5$.
- **GCN** [17]:
  - `hidden1`: 64
  - `early_stopping`: $a \in \{40, 100, 200\}$
  - `epochs`: 2000

  In addition, we disabled the default feature normalization in the official implementation, as the feature vectors in this benchmark have already been normalized, and we found the default normalization method hurts the performance significantly. We report the best performance, for $a = 40$.
- **GCN-Cheby** [17]:
  - `hidden1`: 64
  - `max_degree`: 2
  - `early_stopping`: 40
  - `epochs`: 2000

  We also disabled the default feature normalization in the official implementation for this baseline.
- **GraphSAGE** [11]:
  - `hid_units`: $a \in \{64, 128\}$
  - `lr`: $b \in \{0.1, 0.7\}$
  - `epochs`: 500

  We report the performance with $a = 128, b = 0.1$.
- **MixHop** [1]:
  - `hidden_dims_csv`: $a \in \{64, 192\}$
  - `adj_pows`: 0, 1, 2

  We report the performance with $a = 192$.
- **GAT** [36]:
  - `hid_units`: 8

  We also disabled the default feature normalization in the official implementation for this baseline.
- **MLP**
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: ReLU
  - Dropout Rate: 0.5

Real Benchmark (except Cora-Full) Tuning For each real benchmark in Table 5 (except **Cora-Full**), we perform hyperparameter tuning (see values below) and report the best performance of each method on the validation set. So, for each method, its performance on different benchmarks can be reported from different hyperparameters. We test the following command-line arguments for each baseline method:

- **$\mathbf{H_2}$GCN-1 & $\mathbf{H_2}$GCN-2**:
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: $\{$`ReLU, None`$\}$
  - Dropout Rate: $\{0, 0.5\}$
  - L2 Regularization Weight: $\{$`1e-5, 5e-4`$\}$
- **GCN** [17]:
  - `hidden1`: 64
  - `early_stopping`: $\{40, 100, 200\}$
  - `epochs`: 2000
- **GCN-Cheby** [17]:
  - `hidden1`: 64
  - `weight_decay`: $\{$`1e-5, 5e-4`$\}$
  - `max_degree`: 2
  - `early_stopping`: $\{40, 100, 200\}$
  - `epochs`: 2000
- **GraphSAGE** [11]:
  - `hid_units`: 64
  - `lr`: $\{0.1, 0.7\}$
  - `epochs`: 500
- **MixHop** [1]:
  - `hidden_dims_csv`: $\{64, 192\}$
  - `adj_pows`: 0, 1, 2
- **GAT** [36]:
  - `hid_units`: 8
- **MLP**
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: $\{$`ReLU, None`$\}$
  - Dropout Rate: $\{0, 0.5\}$

For **GCN+JK**, **GCN-Cheby+JK** and **GraphSAGE+JK**, we enhanced the corresponding base model with jumping knowledge (JK) connections using JK-Concat [38] *without* changing the number of layers or other hyperparameters for the base method.

Cora Full Benchmark Tuning The number of class labels in Cora-Full are many more compared to the other benchmarks (Table 5), which leads to a significant increase in the size of training parameters for each model. Therefore, we need to re-tune the hyperparameters, especially the regularization weights and learning rates, in order to get reasonable performance. We test the following command-line arguments for each baseline method:

- **$\mathbf{H_2}$GCN-1 & $\mathbf{H_2}$GCN-2**:
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: $\{$`ReLU, None`$\}$
  - Dropout Rate: $\{0, 0.5\}$
  - L2 Regularization Weight: $\{$`1e-5, 1e-6`$\}$
- **GCN** [17]:

  - `hidden1`: 64
  - `early_stopping`: $\{40, 100, 200\}$
  - `weight_decay`: {`5e-5, 1e-5, 1e-6`}
  - `epochs`: 2000
- **GCN-Cheby** [17]:
  - `hidden1`: 64
  - `weight_decay`: {`5e-5, 1e-5, 1e-6`}
  - `max_degree`: 2
  - `early_stopping`: $\{40, 100, 200\}$
  - `epochs`: 2000
- **GraphSAGE** [11]:
  - `hid_units`: 64
  - `lr`: 0.7
  - `epochs`: 2000
- **MixHop** [1]:
  - `adj_pows`: 0, 1, 2
  - `hidden_dims_csv`: $\{64, 192\}$
  - `l2reg`: {`5e-4, 5e-5`}
- **GAT** [36]:
  - `hid_units`: 8
  - `l2_coef`: {`5e-4, 5e-5, 1e-5`}
- **MLP**
  - Dimension of Feature Embedding p: 64
  - Non-linearity Function σ: {`ReLU, None`}
  - Dropout Rate: $\{0, 0.5\}$
  - L2 Regularization Weight: `1e-5`
  - Learning Rate: 0.05

For **GCN+JK**, **GCN-Cheby+JK** and **GraphSAGE+JK**, we enhanced the corresponding base model with jumping knowledge (JK) connections using JK-Concat [38] *without* changing the number of layers or other hyperparameters for the base method.

G Synthetic Datasets: Details

G.1 Data Generation Process & Setup

Synthetic graph generation We generate synthetic graphs with various heterophily levels by adopting an approach similar to [1, 16]. In general, the synthetic graphs are generated by a modified preferential attachment process [3]: The number of class labels $|\mathcal{Y}|$ in the synthetic graph is prescribed. Then, starting from a small initial graph, new nodes are added into the graph one by one, until the number of nodes $|\mathcal{V}|$ has reached the preset level. The probability p_{uv} for a newly added node u in class i to connect with an existing node v in class j is proportional to both the class compatibility H_{ij} between class i and j, and the degree d_v of the existing node v. As a result, the degree distribution for the generated graphs follow a power law, and the heterophily can be controlled by class compatibility matrix $\mathbf{H}$. Table 3 shows an overview of these synthetic benchmarks, and more detailed statistics can be found in Table G.1.

Node features & classes Nodes are assigned randomly to each class during the graph generation. Then, in each synthetic graph, the feature vectors of nodes in each class are generated by sampling feature vectors of nodes from the corresponding class in a real benchmark (e.g., Cora [30, 39] or `ogbn-products` [13]): We first establish a class mapping $\psi : \mathcal{Y}_s \to \mathcal{Y}_b$ between classes in the synthetic graph $\mathcal{Y}_s$ to classes in an existing benchmark $\mathcal{Y}_b$. The only requirement is that the class size in the existing benchmark is larger than that of the synthetic graph so that an injection between nodes from both classes can be established, and the feature vectors for the synthetic graph can be sampled accordingly. For `syn-products`, we further restrict the feature sampling to ensure that nodes in the training, validation and test splits are only mapped to nodes in the corresponding splits in the benchmark. This process respects the data splits used in `ogbn-products`, which are more realistic and challenging than random splits [13]. For simplicity, in our synthetic benchmarks, all the classes (5 for `syn-cora` and 10 for `syn-products` – Table G.1) are of the same size.

Table G.1: Statistics for Synthetic Datasets

Benchmark Name	`syn-cora`	`syn-products`
# Nodes	1490	10000
# Edges	2965 to 2968	59640 to 59648
# Classes	5	10
Features	`cora` [30, 39]	`ogbn-products` [13]
Homophily h	[0, 0.1, . . . , 1]	[0, 0.1, . . . , 1]
Degree Range	1 to 94	1 to 336
Average Degree	3.98	11.93

Experimental setup For each heterophily ratio h of each benchmark, we independently generate 3 different graphs. For `syn-cora` and `syn-products`, we randomly partition 25% of nodes into training set, 25% into validation and 50% into test set. All methods share the same training, partition and test splits, and the average and standard derivation of the performance values under the 3 generated graphs are reported as the performance under each heterophily level of each benchmark.

G.2 Detailed Results on Synthetic Benchmarks

Tables G.2 and G.3 give the results on `syn-cora` and `syn-products` shown in Figure 2 of the main paper (§ 5.1). Table G.4 provides the detailed results of the ablation studies that we designed in order to investigate the significance of our design choices, and complements Fig. 3 in § 5.1.

Table G.2: `syn-cora` (Fig. 2a): Mean accuracy and standard deviation per method and synthetic dataset (with different homophily ratio h). Best method highlighted in gray.

h	**0.00**	**0.10**	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**	**1.00**
H$_2$GCN-1	77.40±0.89	76.82±1.30	73.38±0.95	75.26±0.56	75.66±2.19	80.22±1.35	83.62±0.82	88.14±0.31	91.63±0.77	95.53±0.61	99.06±0.27
H$_2$GCN-2	77.85±1.63	76.87±0.43	74.27±1.30	74.41±0.43	76.33±1.35	79.60±0.48	84.43±1.89	88.28±0.66	92.39±1.34	95.97±0.59	100.00±0.00
GraphSAGE	75.97±1.94	72.89±2.42	70.56±1.42	71.81±0.67	72.04±1.68	76.55±0.81	81.25±1.04	85.06±0.51	90.78±1.02	95.08±1.16	99.87±0.00
GCN-Cheby	74.23±0.54	68.10±1.75	64.70±1.17	66.71±1.63	68.14±1.56	73.33±2.05	78.88±0.21	84.92±1.03	90.92±1.62	95.97±1.07	100.00±0.00
MixHop	62.64±1.16	58.93±2.84	60.89±1.20	65.73±0.41	67.87±4.01	70.11±0.34	79.78±1.92	84.43±0.94	91.90±2.02	96.82±0.08	100.00±0.00
GCN	33.65±1.68	37.14±4.60	42.82±1.89	51.10±0.77	56.91±2.56	66.22±1.04	77.32±1.17	84.52±0.54	91.23±1.29	96.11±0.82	100.00±0.00
GAT	30.16±1.32	33.11±1.20	39.11±0.28	48.81±1.57	55.35±2.35	64.52±0.47	76.29±1.83	84.03±0.97	90.92±1.51	95.88±0.21	100.00±0.00
MLP	72.75±1.51	74.85±0.76	74.05±0.69	73.78±1.14	73.33±0.34	74.81±1.90	73.42±1.07	71.72±0.62	72.26±1.53	72.53±2.77	73.65±0.41

Table G.3: `syn-products` (Fig. 2b): Mean accuracy and standard deviation per method and synthetic dataset (with different homophily ratio h). Best method highlighted in gray.

h	**0.00**	**0.10**	**0.20**	**0.30**	**0.40**	**0.50**	**0.60**	**0.70**	**0.80**	**0.90**	**1.00**
H$_2$GCN-1	82.06±0.24	78.39±1.56	79.37±0.21	81.10±0.22	84.25±1.08	88.15±0.28	92.39±0.06	95.69±0.19	98.09±0.23	99.63±0.13	99.93±0.01
H$_2$GCN-2	83.37±0.38	80.03±0.84	81.09±0.41	82.79±0.49	86.73±0.66	90.75±0.43	94.81±0.27	97.67±0.18	99.13±0.05	99.89±0.08	99.99±0.01
GraphSAGE	77.66±0.72	74.04±1.07	75.29±0.82	76.39±0.24	80.49±0.96	84.51±0.51	89.51±0.29	93.61±0.52	96.66±0.19	98.78±0.11	99.63±0.08
GCN-Cheby	84.35±0.62	76.95±0.30	77.07±0.49	78.43±0.73	85.09±0.29	89.66±0.53	94.99±0.34	98.26±0.11	99.58±0.11	99.93±0.06	100.00±0.00
MixHop	15.39±1.38	11.91±1.17	14.03±1.70	14.92±0.56	17.04±0.40	18.90±1.49	19.47±5.21	21.15±2.28	24.16±3.19	23.21±5.30	25.09±5.08
GCN	56.44±0.59	51.51±0.56	54.97±0.66	64.90±0.90	76.25±0.04	86.43±0.58	93.35±0.28	97.61±0.24	99.33±0.08	99.93±0.01	99.99±0.01
GAT	27.39±2.47	21.49±2.25	37.27±3.99	44.46±0.68	51.86±8.52	69.42±5.30	85.36±3.67	93.52±1.93	98.84±0.12	99.87±0.06	99.98±0.02
MLP	68.63±0.58	68.20±1.20	68.85±0.73	68.65±0.18	68.37±0.85	68.70±0.61	68.21±0.93	68.72±1.11	68.10±0.54	68.36±1.42	69.08±1.03

Table G.4: Ablation studies of H$_2$GCN to show the significance of designs D1-D3 (Fig. 3(a)-(c)): Mean accuracy and standard deviation per method on the `syn-products` networks.

Design	**h**	**0.00**	**0.10**	**0.20**	**0.30**	**0.40**	**0.50**
D1-D3	**[S0 / K2] H$_2$GCN-1**	82.06±0.24	78.39±1.56	79.37±0.21	81.10±0.22	84.25±1.08	88.15±0.28
D3	**H$_2$GCN-2**	83.37±0.38	80.03±0.84	81.09±0.41	82.79±0.49	86.73±0.66	90.75±0.43
D1	**[NS0] $\mathbf{N_1 + N_2}$**	52.72±0.13	41.65±0.18	46.11±0.86	58.16±0.79	71.10±0.54	82.19±0.40
D1	**[NS1] Only $\mathbf{N_1}$**	40.35±0.58	35.17±0.92	40.35±0.92	52.45±0.85	65.62±0.56	76.05±0.38
D1, D2	**[S1 / N2] w/o** $\bar{N}_2$	79.65±0.27	76.08±0.76	76.46±0.21	77.29±0.46	79.81±0.88	83.56±0.22
D2	**[N1] w/o** $\bar{N}_1$	72.27±0.55	73.05±1.23	75.81±0.67	76.83±0.72	80.49±0.72	82.91±0.44
D2	**[N0] w/o 0-hop neighb. (ego)**	63.55±0.46	46.73±0.42	42.29±0.55	48.20±0.59	61.22±0.35	75.15±0.27
D3	**[K0] No Round-0**	75.63±0.19	61.99±0.57	56.36±0.56	61.27±0.71	73.33±0.88	84.51±0.50
D3	**[K1] No Round-1**	75.75±0.90	75.65±0.73	79.25±0.18	81.19±0.33	84.64±0.35	88.46±0.60
D3	**[R2] Only Round-2**	73.11±1.01	62.47±1.35	59.99±0.43	64.37±1.14	75.43±0.70	86.02±0.79
§ D.2	**Non-linear H$_2$GCN-2 (§ D.2)**	82.23±0.25	78.78±1.04	80.47±0.15	82.08±0.10	85.89±0.53	89.78±0.11

Design	**h**	**0.60**	**0.70**	**0.80**	**0.90**	**0.99**	**1.00**
D1, D3	**[S0 / K2] H$_2$GCN-1**	92.39±0.06	95.69±0.19	98.09±0.23	99.63±0.13	99.88±0.06	99.93±0.01
D3	**H$_2$GCN-2**	94.81±0.27	97.67±0.18	99.13±0.05	99.89±0.08	99.98±0.00	99.99±0.01
D1	**[NS0] $\mathbf{N_1 + N_2}$**	90.39±0.54	95.25±0.06	98.27±0.13	99.69±0.03	99.98±0.02	100.00±0.00
D1	**[NS1] Only $\mathbf{N_1}$**	84.41±0.44	90.15±0.27	95.21±0.34	97.71±0.06	99.56±0.11	99.49±0.11
D1, D2	**[S1 / N2] w/o** $\bar{N}_2$	87.39±0.33	91.08±0.50	94.36±0.32	97.01±0.40	98.79±0.23	98.71±0.15
D2	**[N1] w/o** $\bar{N}_1$	87.24±0.21	92.55±0.50	95.64±0.19	98.71±0.13	99.73±0.12	99.83±0.06
D2	**[N0] w/o 0-hop neighb. (ego)**	86.08±0.58	93.03±0.29	97.45±0.09	99.45±0.06	99.98±0.02	99.98±0.03
D3	**[K0] No Round-0**	92.42±0.13	96.81±0.11	99.09±0.27	99.89±0.01	100.00±0.00	100.00±0.00
D3	**[K1] No Round-1**	93.05±0.23	97.17±0.36	99.06±0.09	99.89±0.08	99.97±0.02	99.97±0.01
D3	**[R2] Only Round-2**	93.79±0.28	97.88±0.18	99.38±0.12	99.89±0.05	100.00±0.00	100.00±0.00
§ D.2	**Non-linear H$_2$GCN-2**	93.68±0.50	96.73±0.23	98.55±0.06	99.74±0.05	99.96±0.04	99.93±0.03

H Real Datasets: Details

Datasets In our experiments, we use the following real-world datasets with varying levels of homophily ratios h. Some network statistics are given in Table 5.

- **Texas, Wisconsin and Cornell** are graphs representing links between web pages of the corresponding universities, originally collected by the CMU WebKB project. We used the preprocessed version in [26]. In these networks, nodes are web pages, which are classified into 5 categories: course, faculty, student, project, staff.
- **Squirrel and Chameleon** are subgraphs of web pages in Wikipedia discussing the corresponding topics, collected by [29]. For the classification task, we utilize the class labels generated by [26], where the nodes are categorized into 5 classes based on the amount of their average traffic.
- **Actor** is a graph representing actor co-occurrence in Wikipedia pages, processed by [26] based on the film-director-actor-writer network in [35]. We also use the class labels generated by [26].
- **Cora, Pubmed and Citeseer** are citation graphs originally introduced in [30, 22], which are among the most widely used benchmarks for semi-supervised node classification [31, 13]. Each node is assigned a class label based on the research field. These datasets use a bag of words representation as the feature vector for each node.
- **Cora Full** is an extended version of Cora, introduced in [4, 31], which contain more papers and research fields than Cora. This dataset also uses a bag of words representation as the feature vector for each node.

Data Limitations As discussed in [31, 13], Cora, Pubmed and Citeseer are widely adopted as benchmarks for semi-supervised node classification tasks; however, all these benchmark graphs display strong homophily, with edge homophily ratio $h \geq 0.7$. As a result, the wide adaptation of these benchmarks have masked the limitations of the homophily assumption in many existing GNN models. Open Graph Benchmark is a recent effort of proposing more challenging, realistic benchmarks with improved data quality comparing to the existing benchmarks [13]. However, with respect to homophily, we found that the proposed OGB datasets display homophily $h > 0.5$.

In our synthetic experiments (§ G), we used `ogbn-products` from this effort to generate higher quality synthetic benchmarks while varying the homophily ratio h. In our experiments on real datasets, we go beyond the typically-used benchmarks (Cora, Pubmed, Citeseer) and consider benchmarks with strong heterophily (Table 5). That said, these datasets also have limitations, including relatively small sizes (e.g., WebKB benchmarks), artificial classes (e.g., Squirrel and Chameleon have class labels based on ranking of page traffic), or unusual network structure (e.g., `Squirrel` and `Chameleon` are dense, with many nodes sharing the same neighbors — cf. § 5.2). We hope that this paper will encourage future work on more diverse datasets with different levels of homophily, and lead to higher quality datasets for benchmarking GNN models in the heterophily settings.