# *SlenderGNN*: ACCURATE, ROBUST, AND INTERPRETABLE GNN, AND THE REASONS FOR ITS SUCCESS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Can we design a GNN that is accurate and interpretable at the same time? Could it also be robust to handle the case of homophily, heterophily, or even noisy edges without network effects? We propose *SlenderGNN* that has all desirable properties: (a) *accurate*, (b) *robust*, and (c) *interpretable*. For the reasons of its success, we had to dig deeper: The result is our GNNLIN framework which highlights the fundamental differences among popular GNN models (e.g., feature combination, structural normalization, etc.) and thus reveals the reasons for the success of our *SlenderGNN*, as well as the reasons for occasional failures of other GNN variants. Thanks to our careful design, *SlenderGNN* passes all the 'sanity checks' we propose, and it achieves the highest overall accuracy on *9* real-world datasets of both homophily and heterophily graphs, when compared against *10* recent GNN models. Specifically, *SlenderGNN* exceeds the accuracy of linear GNNs and matches or exceeds the accuracy of nonlinear models with *up to* $64\times$ fewer parameters.

## 1 INTRODUCTION

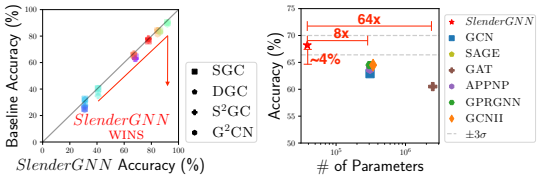
Graph neural networks (GNN) (Defferrard et al., 2016; Kipf & Welling, 2017; Hamilton et al., 2017; Gilmer et al., 2017) are popular methods for various graph-based tasks such as node classification, clustering, or link prediction. GNNs learn low-dimensional representations of nodes by combining the information of node features and a graph structure, providing rich semantic information to solve each task by end-to-end training. There are a huge number of recent variants of GNNs - which one should a practitioner use? Which are the strong and weak points of each variant? Could we design a variant, that has all the strong points, and none of the weak ones?

In response, we propose *SlenderGNN*, a new linear GNN for node classification on both homophily and heterophily graphs, and even on graphs with no network effects. As we elaborate in Section 4, *SlenderGNN* is based on several, careful design decisions (like inclusion of structural information, feature concatenation, etc.) to combine the strengths of prior GNNs, while remedying their pitfalls. As a result, *SlenderGNN* shows notable advantages compared to both linear and nonlinear GNNs:

- *Accurate* on semi-supervised learning, on both real-world and synthetic datasets, almost always winning or tying in the first place (see Table 2 and Table 3).
- *Robust*, being able to handle datasets with homophily; with heterophily; with no network effects; even without tuning any hyperparameters on the propagator function.
- *Slender*, using a few, carefully chosen features, and imposing sparsity on top of that.
- *Interpretable*, thanks to being linear, and thanks to the sparsity of features that we impose (see Figure 1c for details).

This paper has two additional contributions that shed light to the strong and weak points of various GNN models:

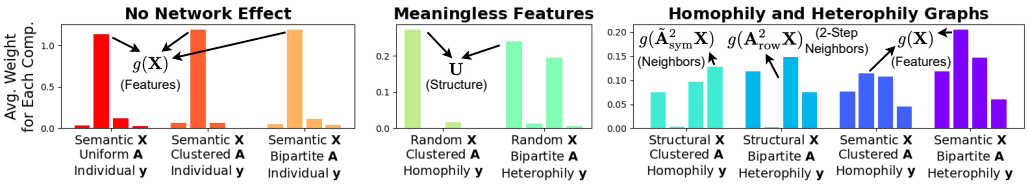
- GNNLIN framework: Using linearization, GNNLIN encompasses recent GNN models and highlights the similarities and differences among them – see Table 1 (and Section 3).
- *Sanity checks*: Based on GNNLIN, we devise diverse scenarios (homophily, heterophily, block-communities, bipartite-graph communities, etc), which easily reveal the strong and weak points of each GNN variant – see Figure 1b and Table 2 (and Section 5).




(a) ***SlenderGNN* performs best** compared with (`left`) linear GNNs and (`right`) state-of-the-art nonlinear GNNs.

| Graph | Feature | LR | SGC | DGC | $S^2$GC | $G^2$CN | SlenderGNN |
|---|---|---|---|---|---|---|---|
| Noisy | Semantic | ✓ | | – | ✓ | | ✓ |
| Homophily | Noisy | | – | | | | ✓ |
| Heterophily | Noisy | | | | | | ✓ |
| Homophily | Structural | – | ✓ | ✓ | ✓ | ✓ | ✓ |
| Heterophily | Structural | – | ✓ | | – | ✓ | ✓ |
| Homophily | Semantic | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Heterophily | Semantic | ✓ | ✓ | | – | | ✓ |

(b) **Only *SlenderGNN* wins all cases.**




(c) ***SlenderGNN* is interpretable**: The correct factor (structure, node features, etc) gets the highest weight.

Figure 1: ***SlenderGNN* outperforms baselines, passes all sanity checks, and is interpretable.** (a) (`left`) We plot accuracy of linear GNNs against *SlenderGNN* for 9 real-world graphs. Points (with datasets as colors) are below the line for all cases, showing that *SlenderGNN* wins. (`right`) We plot accuracy and the number of parameters for *SlenderGNN* and nonlinear GNNs for the Penn94 dataset. *SlenderGNN* is about $4\%$ more accurate with $8\times$ fewer parameters compared with the best competitor. (b) We compare *SlenderGNN* and existing linear GNNs with respect to graph properties: ✓ and – represent accuracy of $\geq 80\%$ and $\geq 60\%$, respectively, in our sanity checks (in Table 2). (c) We show the weights learned by *SlenderGNN* for graphs with different mutual information between variables (labels and factors). *SlenderGNN* assigns large weights to the correct factors. For example, in graphs having no network effects (shown on `left`), the highest weights are assigned to the node features $g(\mathbf{X})$, ignoring structure-based features $\mathbf{U}$, $g(\mathbf{A}^2_{\text{row}}\mathbf{X})$, and $g(\tilde{\mathbf{A}}^2_{\text{sym}}\mathbf{X})$.

**Reproducibility:** Our code is available at https://bit.ly/3fhWJfK along with our datasets for 'sanity checks' and real-world datasets of homophily and heterophily graphs.

## 2 Problem Definition and Related Works

We introduce the problem definition of semi-supervised node classification, symbols frequently used in this paper, and related works on graph neural networks (GNN).

**Problem definition** We define the problem of semi-supervised node classification as follows:

- **Given** An undirected graph $G = (\mathbf{A}, \mathbf{X})$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is an adjacency matrix, $\mathbf{X} \in \mathbb{R}^{n\times d}$ is a node feature matrix, $n$ is the number of nodes, and $d$ is the number of features
- **Given** Labels $\mathbf{y} \in \{1, \cdots, c\}^m$ for $m$ nodes, where $m \ll n$, and $c$ is the number of classes.
- **Predict** the unknown classes of $n - m$ test nodes in $G$.

We use the following symbols to represent modified adjacency matrices. $\tilde{\mathbf{A}} = \mathbf{A}+\mathbf{I}$ is the adjacency matrix with self-loops. $\tilde{\mathbf{D}} = \tilde{\mathbf{A}}\mathbf{1}_{n\times 1}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$, where $\mathbf{1}_{n\times 1}$ is the matrix of size $n \times 1$ filled with ones. $\tilde{\mathbf{A}}_{\text{sym}} = \tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}$ is the symmetrically normalized $\tilde{\mathbf{A}}$. Similarly, $\mathbf{A}_{\text{sym}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ is also a symmetrically normalized $\mathbf{A}$ without self-loops. There are other types of normalization $\mathbf{A}_{\text{row}} = \mathbf{D}^{-1}\mathbf{A}$ and $\mathbf{A}_{\text{col}} = \mathbf{A}\mathbf{D}^{-1}$ (and accordingly $\tilde{\mathbf{A}}_{\text{row}}$ and $\tilde{\mathbf{A}}_{\text{col}}$), which we call row and column normalization, respectively, based on the direction of normalization.

**Graph neural networks** There are a massive number of GNNs proposed with different motivations and insights. Recent surveys (Zhou et al., 2020; Wu et al., 2021) on GNNs introduce a way to categorize existing models into a few groups including spectral models (Defferrard et al., 2016; Kipf & Welling, 2017), sampling-based models (Hamilton et al., 2017; Ying et al., 2018), attention-based models (Velickovic et al., 2018; Kim & Oh, 2021; Brody et al., 2022), and deep models with residual connections (Li et al., 2019; Chen et al., 2020). GNNs are often fused with graphical inference for fully exploiting observed labels at test time (Yoo et al., 2019; Huang et al., 2021). Decoupled GNNs

separate the two major functionalities of GNNs: node-wise feature transformation and feature propagation through the graph (Klicpera et al., 2019a;b; Chien et al., 2021). This allows one to propagate node-level information to distant neighbors without causing the oversmoothing issue.

**Linear graph neural networks** Linear GNNs have been studied widely in literature to improve the robustness and generalizability of GNNs. Wu et al. (2019) proposed SGC, which removes the nonlinear activation functions from GCN (Kipf & Welling, 2017) and reduces the propagator function to a simple matrix multiplication. Wang et al. (2021) and Zhu & Koniusz (2021) improved SGC by manually adjusting the strength of self-loops with hyperparameters, which allows one to increase the number of propagation steps. Li et al. (2022) proposed $G^2$CN, which improves the performance of DGC (Wang et al., 2021) on heterophily graphs by combining multiple propagation settings (i.e. bandwidths). The main limitation of such approaches is the high complexity of propagator functions, which impairs robustness and interpretability even with the linearity with given features.

**Spectral analysis of GNNs** The problem to compare the propagator functions of different GNNs is related to the spectral analysis (Balcilar et al., 2021; Zhu & Koniusz, 2021; Li et al., 2022), which understands different graph convolution operators in the spectral view. However, our linearization-based approach has two notable advantages. First, linearization is generally applicable even to GNN models whose feature transformation and propagation are mixed. Second, linearization gives a clear view on how the features interact with the graph structure and make predictions. To the best of our knowledge, linearization-based comparison between GNNs has not been studied in literature.

## 3 Proposed Framework: GnnLin

We propose GnnLin, a framework that generalizes existing linear graph neural network (GNN) and provides a way of systematic linearization for existing GNN models (see Figure 2). Our observations from linearization motivate us to propose *SlenderGNN*, a new GNN model elaborated in Section 4.

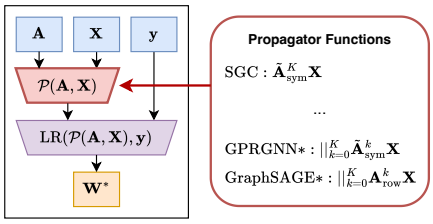



Figure 2: Linearization through GnnLin.

### 3.1 Definition of GnnLin

**Definition 1** (LR)**.** *Given a feature $\mathbf{X} \in \mathbb{R}^{n \times d}$ and a label $\mathbf{y} \in \mathbb{R}^m$, where $m \leq n$ is the number of observations, let $\mathbf{Y} \in \mathbb{R}^{m \times c}$ be the one-hot representation of $\mathbf{y}$, and $y_{ij}$ be the $(i, j)$-th element in $\mathbf{Y}$. Then, logistic regression (LR) finds an optimal weight matrix $\mathbf{W}^* \in \mathbb{R}^{d \times c}$ as follows:*

$$\mathrm{LR}(\mathbf{X}, \mathbf{y}) = \arg\max_{\mathbf{W}} \sum_{i=1}^{l} \sum_{j=1}^{c} y_{ij} \log \hat{y}_{ij} \quad \text{where} \quad \hat{y}_{ij} = \frac{\exp(\mathbf{w}_{\cdot j}^\top \mathbf{x}_i)}{\sum_{k=1}^{c} \exp(\mathbf{w}_{\cdot k}^\top \mathbf{x}_i)}, \tag{1}$$

*and $\mathbf{w}_{\cdot j}$ is the $j$-th column of $\mathbf{W}$. We omit the bias term for brevity without loss of generality.*

**Definition 2** (Linearized GNN)**.** *Given a graph $G = (\mathbf{A}, \mathbf{X})$, let $f(\cdot; \theta)$ be a node classifier function to predict the labels of all nodes in $G$ as $\hat{\mathbf{y}} = f(\mathbf{A}, \mathbf{X}; \theta)$, where $\theta$ is the set of learnable parameters. Then, $f$ is* linearized *if $\theta = \{\mathbf{W}\}$ and an optimal weight matrix $\mathbf{W}^* \in \mathbb{R}^{h \times c}$ is given as*

$$\mathbf{W}^* = \mathrm{LR}(\mathcal{P}(\mathbf{A}, \mathbf{X}), \mathbf{y}), \tag{2}$$

*where $\mathcal{P}$ is a feature propagator function that is linear with $\mathbf{X}$ and contains no learnable parameters, and $\mathcal{P}(\mathbf{A}, \mathbf{X}) \in \mathbb{R}^{n \times h}$. We ignore the bias term for brevity without loss of generality.*

**Definition 3** (GnnLin)**.** *Let $f(\cdot; \theta)$ be a (nonlinear) GNN.* GnnLin *is to represent $f$ as a linearized GNN by replacing all (nonlinear) activation functions in $f$ with the identity function and deriving a variant $f'$ that is at least as expressive as $f$ but contains no parameters in $\mathcal{P}$.*

GnnLin represents the characteristic of a GNN as the linear feature propagation function $\mathcal{P}$, which transforms raw features $\mathbf{X}$ by utilizing $\mathbf{A}$. Lemma 1 shows that GnnLin generalizes existing linear GNNs. Logistic regression is also represented by GnnLin with the identity $\mathcal{P}(\mathbf{A}, \mathbf{X}) = \mathbf{X}$.

**Lemma 1.** *Existing linear models such as SGC (Wu et al., 2019), DGC (Wang et al., 2021), $S^2$GC (Zhu & Koniusz, 2021), and $G^2$CN (Li et al., 2022) are represented by* GnnLin.

*Proof.* The proof is given in Appendix A.1. ■

Table 1: **GNNLIN encompasses popular GNN models with various types.** The '*' symbol in the name of a model represents that it is linearized, and '**' represents incomplete linearization because of the attention function (details in Section 3.2). The result of comprehensive linearization provides us observations and insights on the core mechanisms of GNNs (details in Section 3.3).

| Model | Type | Propagator function $\mathcal{P}(\mathbf{A}, \mathbf{X})$ | Hyperparameters |
|---|---|---|---|
| LR | Linear | $\mathbf{X}$ | - |
| SGC | Linear | $\tilde{\mathbf{A}}_{\text{sym}}^{K}\mathbf{X}$ | $K$ |
| DGC | Linear | $[(1-T/K)\mathbf{I} + (T/K)\tilde{\mathbf{A}}_{\text{sym}}]^{K}\mathbf{X}$ | $K, T$ |
| $\text{S}^2$GC | Linear | $\sum_{k=1}^{K}(\alpha\mathbf{I} + (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^{k})\mathbf{X}$ | $K, \alpha$ |
| $\text{G}^2$CN | Linear | $\Vert_{i=1}^{N}[\mathbf{I} - (T_i/K)((b_i-1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]^{K}\mathbf{X}$ | $K, N, T_i, b_i$ |
| PPNP* | Decoupled | $(\mathbf{I} - (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}})^{-1}\mathbf{X}$ | $\alpha$ |
| APPNP* | Decoupled | $[\sum_{k=0}^{K-1}\alpha(1-\alpha)^{k}\tilde{\mathbf{A}}_{\text{sym}}^{k} + (1-\alpha)^{K}\tilde{\mathbf{A}}_{\text{sym}}^{K}]\mathbf{X}$ | $K, \alpha$ |
| GDC* | Decoupled | $\tilde{\mathbf{S}}_{\text{sym}}\mathbf{X}$ where $\mathbf{S} = \text{sparse}_{\epsilon}(\sum_{k=0}^{\infty}(1-\alpha)^{k}\tilde{\mathbf{A}}_{\text{sym}}^{k})$ | $\alpha, \epsilon$ |
| GPR-GNN* | Decoupled | $\Vert_{k=0}^{K}\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}$ | $K$ |
| ChebNet* | Coupled | $\Vert_{k=0}^{K-1}\mathbf{A}_{\text{sym}}^{k}\mathbf{X}$ | $K$ |
| GCN* | Coupled | Same as SGC | $K$ |
| SAGE* | Coupled | $\Vert_{k=0}^{K}\mathbf{A}_{\text{row}}^{k}\mathbf{X}$ | $K$ |
| GCNII* | Coupled | $\Vert_{k=0}^{K-2}\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X} \Vert ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^{K} + \alpha\tilde{\mathbf{A}}_{\text{sym}}^{K-1})\mathbf{X}$ | $K, \alpha$ |
| GAT** | Attention | $\prod_{k=1}^{K}[\text{diag}(\mathbf{X}\mathbf{w}_{k,1})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\text{diag}(\mathbf{X}\mathbf{w}_{k,2})]\mathbf{X}$ | $K, \mathbf{w}_{k,1}, \mathbf{w}_{k,2}$ |
| DA-GNN** | Attention | $\sum_{k=0}^{K}\text{diag}(\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}\mathbf{w})\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}$ | $K, \mathbf{w}$ |

The first few rows of Table 1 show the representation of linear models as GNNLIN. DGC and $\text{S}^2$GC improve SGC by controlling the strength of self-loops with new hyperparameters $T$ and $\alpha$. $\text{G}^2$CN is the latest linear model that achieves highest accuracy, but has the most complex propagator function with many hyperparameters $K$, $N$, $T_i$, and $b_i$. Such an increase of complexity limits the robustness and interpretability of decisions, which are the main advantages of linear models.

### 3.2 Linearization of Existing Models

We apply GNNLIN to represent existing GNNs in the linearized forms in Table 1, which allows us to understand and compare all linear and nonlinear GNNs in the form of propagator functions.

**Decoupled models** PPNP, APPNP (Klicpera et al., 2019a), GDC (Klicpera et al., 2019b), and GPR-GNN (Chien et al., 2021) are decoupled GNNs that separate feature transformation and propagation stages. PPNP runs Personalized PageRank on the node features, and APPNP approximates PPNP with $K$ steps of message propagation. GDC generalizes APPNP by increasing the value of $K$ to $\infty$ and sparsifies the propagator matrix $\mathbf{S}$. GPR-GNN also generalizes APPNP to avoid the usage of $\alpha$ by learning a weight for each component, resulting in the concatenation of multiple different features. We provide details of linearization in Appendix A.2.

**Coupled models** We linearize coupled GNNs including ChebNet (Defferrard et al., 2016), GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), and GCNII (Chen et al., 2020). The linearized version of GCN is the same as SGC, since the motivation of SGC is to linearize GCN for better scalability and robustness. Although their motivations are different, the linearized versions of ChebNet, GraphSAGE, and GCNII are similar to linearized GPR-GNN in that features propagated by different steps are combined by concatenation. We provide the details in Appendix A.3.

**Attention models** Attention-based GNNs (Velickovic et al., 2018; Kim & Oh, 2021; Brody et al., 2022) are a popular category of GNNs to learn the importance of each edge based on $\mathbf{X}$ and learnable parameters. Thus, it is not straightforward to linearize them by the GNNLIN framework: even if we assume the learnable parameters in $\mathcal{P}$ as fixed hyperparameters, $\mathcal{P}$ is at least quadratic with $\mathbf{X}$, since $\mathbf{X}$ participates in computing the new adjacency matrix that is multiplied again with $\mathbf{X}$. Nevertheless, we perform incomplete linearization of attention-based models for completeness, and represent the results with the '**' symbol in Table 1. Detailed processes are given in Appendix A.4.

The linearized version of GAT uses a different adjacency matrix for each step $k$, which is the main difference from other linearized models. It is noteworthy that DA-GNN (Liu et al., 2020), proposed as a decoupled model in their paper, is an attention-based model in our analysis. This is because $\mathbf{X}$ participates in computing the new adjacency matrix, making $\mathcal{P}$ quadratic as in GAT.

### 3.3 Distinguishing Factors and Pain Points

Table 1 shows that many existing GNNs have similar or even identical propagator functions after we linearize them. We present the main factors that make a difference between those models and detect pain points that none of the existing models in Table 1 addresses well.

**Distinguishing Factor 1** (Combination of features)**.** *How should we combine the node features, the immediate neighbors' features, and the $k$-step-away neighbors' features?*

GNNs propagate information by multiplying the feature $\mathbf{X}$ with (a variant of) the adjacency matrix $\mathbf{A}$ multiple times. Three approaches prevail in Table 1: **(1)** no aggregation (SGC, DGC, G$^2$CN, and GAT), **(2)** summation (S$^2$GC, APPNP, and GDC), and **(3)** concatenation (GPR-GNN, GraphSAGE, and GCNII). The simplest approach is to use only the one with $K$ steps (i.e. no aggregation), while the other two approaches aim to combine the information of multiple steps.

**Distinguishing Factor 2** (Modification of adjacency matrices)**.** *How should we normalize or modify the adjacency matrix to better represent the connectivity information?*

The three prevailing choices are **(1)** symmetric vs. row normalization, **(2)** the strength of self-loops, including making zero self-loops, and **(3)** static vs. dynamic adjustment based on the given features. Most models use the symmetric normalization $\tilde{\mathbf{A}}_{\mathrm{sym}}$ with self-loops, but some variants avoid self-loops and use either row normalization $\mathbf{A}_{\mathrm{row}}$ or symmetric one $\mathbf{A}_{\mathrm{sym}}$. Recent models such as DGC, G$^2$CN, and GCNII determine the weight of self-loops with hyperparameters, since strong self-loops allow one to increase the value of $K$ for distant propagation. Finally, attention-based models learn the elements in $\mathbf{A}$ based on node features, making propagator functions quadratic with $\mathbf{X}$.

**Distinguishing Factor 3** (Dealing with heterophily graphs)**.** *How should we consider heterophily if the direct neighbors are likely to have the information of different labels?*

In the case of heterophily, using the features of immediate neighbors can actually hurt performance, and several GNN models do suffer under heterophily. Observed ways to handle it are **(1)** using the square of $\mathbf{A}$ as the main unit, **(2)** learning different weights for different steps, and **(3)** making small or no self-loops in $\mathbf{A}$. The idea of such approaches is to avoid or downplay the effect of immediate (and odd-step-away) neighbors. Self-loops hurt under heterophily, as they force to have information of all intermediate neighbors by acting as implicit summation of transformed features.

**Pain Point 1** (Hyperparameters in propagators)**.** *The hyperparameters in a propagator function $\mathcal{P}$ impairs the interpretability of the weight matrix $\mathbf{W}$, and forces one to re-compute the transformed feature for every new choice during hyperparameter search.*

The last column of Table 1 summarizes the hyperparameters in each $\mathcal{P}$, which make the following limitations in terms of linear models. First, the interpretability of the weight matrix $\mathbf{W}$ is impaired, because it is learned on the transformed feature $\mathcal{P}(\mathbf{A}, \mathbf{X})$ whose meaning changes arbitrarily by the choice of hyperparameters. Second, $\mathcal{P}(\mathbf{A}, \mathbf{X})$ should be computed for each choice of hyperparameters, while it can be easily cached for searching hyperparameters outside $\mathcal{P}$.

**Pain Point 2** (Failure on noisy features)**.** *All models in Table 1 depend on the node feature matrix $\mathbf{X}$, and cannot fully exploit the adjacency matrix $\mathbf{A}$ if the given features are noisy.*

Real-world datasets often contain noisy features, and the graph structure $\mathbf{A}$ plays an essential role in the performance of node classification in such cases. That is, a desirable property for a robust model is to adaptively emphasize important features or disregard noisy ones to maximize its generalization performance. However, the models in Table 1 lack such a functionality.

## 4 Proposed Method - *SlenderGNN*

Based on the insights from the comprehensive linearization, we propose *SlenderGNN*, a novel linear GNN that addresses the limitations of existing GNN models with a simple propagator function. The

propagator function $\mathcal{P}$ of *SlenderGNN* is given as follows:

$$\boxed{\mathcal{P}(\mathbf{A}, \mathbf{X}) = \underbrace{\mathbf{U}}_{\text{Structure}} \parallel \underbrace{g(\mathbf{X})}_{\text{Node features}} \parallel \underbrace{g(\mathbf{A}^2_{\text{row}}\mathbf{X})}_{\text{2-step neighbors}} \parallel \underbrace{g(\tilde{\mathbf{A}}^2_{\text{sym}}\mathbf{X})}_{\text{Neighbors}}} \tag{3}$$

where $g(\cdot)$ is the principal component analysis (PCA) for the orthogonalization of each component, followed by an L2 normalization, and $\mathbf{U} \in \mathbb{R}^{n \times r}$ contains $r$-dimensional structural features derived by running the low-rank support vector decomposition (SVD) on the matrix $\mathbf{A}$.

The goals of *SlenderGNN* are to maximize (a) the robustness to graphs of various properties and (b) the interpretability of learned weights. We emphasize robustness, since in semi-supervised learning, high accuracy on real-world datasets is acquired naturally from robustness. We introduce the *design decisions* D1-D5, that lead us to propose Equation 3 as our approach to achieve our goals. D1 is to achieve both goals, D2 and D3 are for robustness, and D4 and D5 are designed for interpretability. In response to Pain Point 1, Equation 3 is designed to contain no hyperparameters to tune.

**D1: Concatenation-based linear method** The main principle of *SlenderGNN* is to make linear decisions to acquire robustness and interpretability at the same time. Recent works on linear GNNs improve the limited expressiveness of linear decisions by designing more complex $\mathcal{P}$. On the other hand, Table 1 suggests that many GNNs concatenate the features of heterogeneous information, and it leads to combining rich information of a graph structure even remaining the low complexity of $\mathcal{P}$. The concatenation also allows us to ignore meaningless components in the transformed features and to interpret the learned weights as the importances of given components in each dataset.

**D2: Structural features (for robustness)** In response to Pain Point 2 where features are missing, noisy, or useless for classification, then we have to resort to the adjacency matrix $\mathbf{A}$ ignoring $\mathbf{X}$. At the same time, it is not effective to use raw $\mathbf{A}$ in $\mathcal{P}$, considering that it is a large sparse matrix. We thus adopt low-rank SVD with rank $r$ to extract structural features $\mathbf{U}$. The value of $r$ is automatically selected to keep $90\%$ of the energy of $\mathbf{A}$, where the sum of the largest $r$ squared singular values divided by the squared Frobenius norm of $\mathbf{A}$ is approximately $0.9$. When the graph is large, we set $r$ to be $d$ for the size consistency of components (see D5).

**D3: Combining different normalizations (for robustness)** The symmetric normalization $\tilde{\mathbf{A}}_{\text{sym}}$ with self-loops is known to perform well in many cases, but has two limitations. First, the self-loops in $\tilde{\mathbf{A}}^K_{\text{sym}}$ force one to consider all intermediate nodes until the $K$-hop neighbors, even in heterophily graphs where the direct neighbors should be avoided. Second, the neighboring features are rescaled based on the node degrees during an aggregation. As a solution, we combine different normalization methods in feature transformation: $\tilde{\mathbf{A}}_{\text{sym}}$ and $\mathbf{A}_{\text{row}}$, where the row-normalized matrix $\mathbf{A}_{\text{row}}$ does not include self-loops. $\mathbf{A}_{\text{row}}$ effectively summarizes the information of $K$-hop neighbors preserving the scales of features, acquiring better signals on heterophily graphs.

**D4: Keeping the low power (for interpretability)** Existing linear GNNs use a very large value of $K \geq 50$, which determines the power of a (transformed) adjacency matrix $\mathbf{A}$. This is effective in distant propagation of information, but severely limits the interpretability of the learned weight $\mathbf{W}$: for example, one cannot tell where the information comes in a graph when features are transformed with $\mathbf{A}^{50}\mathbf{X}$. Therefore, we propose to keep the small value of $K = 2$ for the interpretability, while taking the information of two-hop neighbors is essential for many cases.

**D5: Orthogonalization and sparsification (for interpretability)** We use two reliable methods to improve the interpretability of *SlenderGNN*: dimensionality reduction by PCA and regularization by group LASSO. First, we run PCA on each component independently to orthogonalize given features and to improve the consistency of learned weights. Second, we apply group LASSO to learn sparse weights on the component level, while preserving the relative magnitude of each element. To make the consistency between components, we force all components to have the same dimensionality by selecting $r$ features from each component when adopting PCA.

## 5 Proposed Sanity Checks on Possible Graph Scenarios

We propose *sanity checks* to evaluate the robustness of GNNs to various scenarios of node classification and to observe their strengths and weaknesses in different settings.

Table 2: ***SlenderGNN* passes all sanity checks.** Accuracy of all models on sanity checks; three groups of scenarios: (left) only features $\mathbf{X}$ are informative; (middle) only connectivity $\mathbf{A}$ is informative; (right) both are informative. The three letters at the second row represent the cases of $\mathbf{X}$, $\mathbf{A}$, and $\mathbf{y}$, respectively: S (semantic $\mathbf{X}$), R (random $\mathbf{X}$), T (structural $\mathbf{X}$), U (uniform $\mathbf{A}$), C (clustered $\mathbf{A}$), B (bipartite $\mathbf{A}$), I (individual $\mathbf{y}$), O (homophily $\mathbf{y}$), and E (heterophily $\mathbf{y}$). The colors ■, ■, ■ represent the top three methods within $2\sigma$ of the best accuracy at each setting, and ■ represents accuracy below $2\sigma$ of the third-best accuracy.

| **Model** | $\mathrm{MI}(\mathbf{X},\mathbf{y})>0$, i.e., $\mathbf{X}$ helps | | | $\mathrm{MI}(\mathbf{A},\mathbf{y})>0$ | | $\mathrm{MI}(\mathbf{X},\mathbf{A},\mathbf{y})>0$, i.e., both $\mathbf{X}$ and $\mathbf{A}$ help | | | |
|---|---|---|---|---|---|---|---|---|---|
| | (S, U, I) Uniform | (S, C, I) Uniform | (S, B, I) Uniform | (R, C, O) Homophily | (R, B, E) Heterophily | (T, C, O) Homophily | (T, B, E) Heterophily | (S, C, O) Homophily | (S, B, E) Heterophily |
| LR | 83.7±0.6 | 83.7±0.6 | 83.7±0.6 | 24.2±0.7 | 24.2±0.7 | 71.4±0.9 | 66.8±2.2 | 83.4±0.6 | 83.4±0.6 |
| SGC | 44.6±9.8 | 43.0±9.0 | 45.1±10. | 64.3±0.7 | 50.2±14. | 87.1±0.6 | 84.3±0.5 | 93.9±0.9 | 91.5±0.5 |
| DGC | 63.8±1.0 | 66.3±0.9 | 65.9±1.2 | 50.5±13. | 26.0±0.9 | 88.6±1.0 | 45.3±1.3 | 96.2±0.4 | 54.0±0.6 |
| $S^2$GC | 79.9±0.6 | 79.8±1.1 | 80.1±1.0 | 38.5±12. | 25.4±0.9 | 88.4±1.0 | 67.9±1.5 | 95.9±0.6 | 78.0±0.5 |
| $G^2$CN | 25.2±0.3 | 25.3±0.1 | 25.0±0.2 | 24.2±1.1 | 25.0±0.1 | 88.5±1.0 | 88.6±1.2 | 24.3±1.1 | 50.7±31. |
| GCN | 36.3±3.5 | 33.2±2.4 | 35.7±3.5 | 46.7±8.0 | 43.7±1.9 | 83.3±1.3 | 72.2±1.7 | 91.2±1.2 | 80.3±3.9 |
| SAGE | 80.3±1.1 | 79.0±1.2 | 79.8±0.7 | 31.1±0.7 | 34.6±2.1 | 83.9±0.8 | 81.3±0.7 | 94.4±0.5 | 94.4±0.9 |
| GCNII | 73.5±1.2 | 73.2±0.9 | 73.6±0.9 | 30.7±0.7 | 27.1±1.3 | 84.2±0.8 | 69.0±1.4 | 90.6±0.9 | 80.4±1.2 |
| APPNP | 66.0±2.6 | 65.4±2.7 | 64.2±1.6 | 30.3±1.2 | 25.2±0.7 | 71.2±4.9 | 43.8±2.0 | 83.2±3.8 | 58.7±4.5 |
| GPR-GNN | 73.4±0.4 | 73.5±1.3 | 73.9±0.7 | 74.6±0.7 | 65.9±2.1 | 89.9±0.6 | 87.6±1.2 | 95.0±1.1 | 91.9±1.1 |
| GAT | 32.7±5.5 | 30.6±3.0 | 33.8±6.8 | 42.6±4.8 | 36.8±5.7 | 64.0±5.7 | 55.6±6.8 | 68.5±7.1 | 67.0±12. |
| ***SlenderGNN*** | 81.0±1.1 | 81.0±0.9 | 81.2±1.0 | 87.1±1.4 | 89.2±1.2 | 88.1±0.5 | 88.9±0.7 | 94.4±0.6 | 93.9±0.5 |

**Graph scenarios** We categorize possible scenarios of node classification based on the characteristics of node features $\mathbf{X}$, a graph structure $\mathbf{A}$, and node labels $\mathbf{y}$. We describe only the main ideas, leaving exact definitions of such scenarios to Appendix B.1.

- **Edges $\mathbf{A}$:** We consider three cases: *uniform* (no communities), *clustered* (block-diagonal), and *bipartite*. The uniform case means that every element $a_{ij}$ is determined independently of the other edges in $\mathbf{A}$. In the clustered case, nodes having common neighbors are likely to make more edges, while it is the opposite in the bipartite case.
- **Labels $\mathbf{y}$:** We consider three cases: *individual* (no network effects), *homophily*, and *heterophily*. In the homophily case, adjacent nodes are likely to have the same label, while it is the opposite in the heterophily case.
- **Features $\mathbf{X}$:** We consider three cases: *random*, *structural*, and *semantic*. The random case means that each feature is determined independently of all other variables. In the structural case, features give information of the graph structure, and in the semantic case, they directly provide useful information for node labels. Unlike the cases of edges and labels, which are mutually exclusive, features can be both structural and semantic at the same time.

**Feasibility** Although there exist a total of 27 combinations for $(\mathbf{X},\mathbf{A},\mathbf{y})$, not all of them are possible to implement; for example, either homophily or heterophily $\mathbf{y}$ is not compatible with uniform $\mathbf{A}$. After removing the infeasible combinations of variables, we categorize the remaining choices based on the mutual information (MI) between variables.

**Implementation** For generating synthetic graphs, we divide all nodes into $c$ clusters, where $c=4$ in our experiments, and determine the edge densities in each cluster and between different clusters based on the property of $\mathbf{A}$. Details on implementation are described in Appendix B.2.

**Results of sanity checks** Table 2 shows the results of sanity checks for our *SlenderGNN* and all baseline models (details of the baselines in Section 6). We run each experiment five times and report the average and standard deviation. We assume four target classes of nodes, and thus the accuracy of random guessing is 25%. Logistic regression (LR) fails when $\mathrm{MI}(\mathbf{X},\mathbf{y})=0$, since it uses only $\mathbf{X}$ for classification. LR shows relatively lower accuracy with structural features, since they are not a perfect indicator for node labels, and they need to be refined through $\mathbf{A}$ for better performance.

It is clear that our *SlenderGNN* is the only approach that passes all sanity checks. The four components in *SlenderGNN* are carefully designed to maximize its robustness for various graph scenarios. Most GNNs work well in the cases (S, C, O) and (T, C, O), where $\mathbf{X}$ is either semantic or structural, $\mathbf{A}$ is clustered, and $\mathbf{y}$ is homophily, since many real-world datasets that recent works on GNNs use

in their experiments follow such assumptions. However, many GNNs fail when a graph is generated with different assumptions, as we summarize as follows:

- **No network effects:** In the cases of (S, ?, I), where ? is a placeholder, only a few models such as SAGE and GPR-GNN perform well. This is because $\mathrm{MI}(\mathbf{A}, \mathbf{y}) = 0$ in such cases, and models are required to focus on raw features $\mathbf{X}$ ignoring $\mathbf{A}$.
- **Meaningless features:** In the cases of (R, C, O) and (R, B, E), models are required to do the opposite since $\mathbf{MI}(\mathbf{X}, \mathbf{y}) = 0$: models should focus on $\mathbf{A}$, ignoring $\mathbf{X}$. Since there is no approach that explicitly uses $\mathbf{A}$, all baselines show low accuracy.
- **Heterophily graphs:** In the cases of (?, B, E), where ? is a placeholder, the labels follow heterophily. Models like $G^2$CN, SAGE, and GPR-GNN perform well in these cases, since they address the heterophily in their designs (details in Section 3.3).

# 6 Experiments

## 6.1 Experimental Settings

**Datasets** We use 5 homophily and 4 heterophily datasets, which were used in previous works on node classification (Chien et al., 2021; Pei et al., 2020). Cora, CiteSeer, and PubMed (Sen et al., 2008; Yang et al., 2016) are homophily citation graphs between research articles. Computers and Photo (Shchur et al., 2018) are homophily Amazon co-purchase graphs between items. Chameleon and Squirrel are heterophily Wikipedia graphs (Rozemberczki et al., 2021), connected by links between pages. Actor (Tang et al., 2009) is a heterophily graph connected by co-occurrence of actors on the same Wikipedia pages. Penn94 (Traud et al., 2012) is a heterophily graph of gender relations in a social network. We make the heterophily graphs undirected, as done in (Chien et al., 2021).

**Competitors** We compare our *SlenderGNN* with various types of graph neural networks (GNN). Logistic regression (LR) is the simplest baseline, which does not use graph information. SGC (Wu et al., 2019), DGC (Wang et al., 2021), $S^2$GC (Zhu & Koniusz, 2021), and $G^2$CN (Li et al., 2022) are linear GNNs that perform as our main competitors. GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GCNII (Chen et al., 2020) are coupled GNN models popular in literature. APPNP (Klicpera et al., 2019a) and GPR-GNN (Chien et al., 2021) are decoupled GNNs, and GAT (Velickovic et al., 2018) is an attention-based model. We perform hyperparameter search based on those reported in their original papers. Refer to Appendix C for details.

**Experimental setup** We perform semi-supervised node classification by dividing all nodes in a graph by the $2.5\%/2.5\%/95\%$ ratio into training, validation, and testing data. We perform five runs of each experiment with different random seeds and report the average and standard deviation. All hyperparameter search and early stopping are done based on validation accuracy for each run.

## 6.2 Experimental Results

**Accuracy on real-world graphs** In Table 3, *SlenderGNN* is compared against linear as well as the state-of-the-art nonlinear GNNs on 9 real-world datasets (5 homophily and 4 heterophily graphs). We report the accuracy in Table 3 where ■, ■, ■ represent top three methods within $2\sigma$ of the best method, and ■ represents accuracy below $2\sigma$ of the third-best method. Our proposed *SlenderGNN* outperforms all the competitors in 2 homophily and 3 heterophily datasets, and shows competitive results in the rest (7 out of 9 times among top 3 methods). Moreover, it is the only model without red cells, which demonstrates the robustness and generality of *SlenderGNN*.

*SlenderGNN* is also slender, where we report the number of parameters in Table 6 in the appendix. Compared with linear models, *SlenderGNN* achieves the best trade-off between the model size and two benefits, namely accuracy and interpretability; compared with nonlinear models, it has much fewer parameters, while performing no worse than them.

**Ablation study** We report the ablation study in Table 4, by removing each of the essential modules of *SlenderGNN*. The results show that *SlenderGNN* is most robust and effective with all the modules.

**Interpretability of learned weights** As shown in Figure 1c, we illustrate the learned weights of *SlenderGNN* in our proposed 'sanity checks', where the ground truths are known. In the graphs without network effects, *SlenderGNN* successfully assigns the highest weights to the node features

Table 3: ***SlenderGNN* is effective on real-world graphs and wins most of the times.** Accuracy on real-world graphs (5 homophily and 4 heterophily graphs) for all the competing models. The colors ■, ■, ■ represent the top three methods within $2\sigma$ of the best accuracy at each setting, and ■ represents accuracy below $2\sigma$ of the third-best accuracy.

| Model | Cora | CiteSeer | PubMed | Computers | Photo | Chameleon | Squirrel | Actor | Penn94 |
|---|---|---|---|---|---|---|---|---|---|
| LR | 51.5±1.2 | 52.9±4.5 | 79.9±0.5 | 73.9±1.2 | 79.3±1.5 | 24.9±1.7 | 26.7±1.9 | 27.8±0.8 | 63.5±0.5 |
| SGC | 76.2±1.1 | 65.8±3.9 | 84.1±0.8 | 83.7±1.6 | 90.1±0.9 | 38.1±4.5 | 33.1±1.0 | 24.6±0.8 | 64.0±1.1 |
| DGC | 77.8±1.4 | 66.1±4.2 | 84.3±0.6 | 83.9±0.7 | 90.4±0.2 | 37.2±3.7 | 29.2±1.2 | 25.2±2.1 | 62.5±0.4 |
| $S^2$GC | 78.3±1.5 | 66.9±4.4 | 84.3±0.3 | 83.1±0.8 | 90.1±0.8 | 34.9±4.9 | 27.6±1.8 | 26.7±1.8 | 63.1±0.5 |
| $G^2$CN | 76.6±1.5 | 64.2±3.3 | 81.4±0.6 | 82.8±1.6 | 88.8±0.5 | 40.7±2.9 | 32.1±1.5 | 24.3±0.5 | O.O.M. |
| GCN | 76.0±1.2 | 65.0±2.9 | 84.3±0.5 | 85.1±0.9 | 91.6±0.5 | 38.5±3.0 | 31.4±1.8 | 26.8±0.4 | 62.9±0.7 |
| SAGE | 74.6±1.3 | 63.7±3.6 | 82.9±0.4 | 83.8±0.5 | 90.6±0.5 | 39.8±4.3 | 27.0±1.3 | 27.8±0.9 | O.O.M. |
| GCNII | 77.8±1.7 | 63.4±3.0 | 84.9±0.8 | 82.3±1.8 | 90.8±0.6 | 30.5±2.5 | 21.9±3.0 | 29.0±1.3 | 64.5±0.5 |
| APPNP | 80.0±0.6 | 67.1±2.8 | 84.6±0.5 | 84.2±1.7 | 92.5±0.3 | 30.9±4.7 | 23.9±3.2 | 26.1±1.0 | 63.7±0.9 |
| GPR-GNN | 78.8±1.3 | 64.2±4.0 | 85.1±0.7 | 85.0±1.0 | 92.6±0.3 | 31.7±4.7 | 26.2±1.6 | 29.5±1.1 | 64.5±0.4 |
| GAT | 78.2±1.2 | 65.8±4.0 | 83.6±0.2 | 85.4±1.4 | 91.7±0.5 | 39.1±4.1 | 28.6±0.6 | 26.4±0.4 | 60.5±0.8 |
| *SlenderGNN* | 77.8±1.1 | 67.1±2.3 | 84.6±0.5 | 86.3±0.7 | 91.8±0.5 | 40.8±3.2 | 31.1±0.7 | 30.9±0.6 | 68.2±0.6 |

Table 4: **Ablation study: all proposed components of *SlenderGNN* are needed.** We evaluate the accuracy of *SlenderGNN* when each of the components is disabled.

| Model | Cora | CiteSeer | PubMed | Computers | Photo | Chameleon | Squirrel | Actor | Penn94 |
|---|---|---|---|---|---|---|---|---|---|
| w/o Sp. Reg. | 77.8±0.6 | 65.0±3.5 | 83.8±0.5 | 85.9±0.8 | 91.7±0.7 | 40.1±3.8 | 30.7±1.0 | 30.1±0.6 | 67.4±0.6 |
| w/o PCA | 74.8±1.5 | 66.0±3.1 | 84.7±0.5 | 84.4±1.1 | 90.3±0.7 | 41.3±2.0 | 31.8±1.1 | 27.3±1.1 | 67.7±0.7 |
| w/o $\mathbf{U}$ | 78.1±1.0 | 67.4±2.5 | 84.4±0.3 | 86.0±0.5 | 92.1±0.4 | 37.5±4.2 | 29.8±0.4 | 31.3±0.5 | 65.8±0.6 |
| *SlenderGNN* | 77.8±1.1 | 67.1±2.3 | 84.6±0.5 | 86.3±0.7 | 91.8±0.5 | 40.8±3.2 | 31.1±0.7 | 30.9±0.6 | 68.2±0.6 |

$g(\mathbf{X})$. When the features are meaningless, it ignores the features and puts most of the attention on the extracted structural features $\mathbf{U}$. In the cases that structural features are useful, but not enough to correctly identify the labels, it gives the highest weights to the propagated features $g(\tilde{\mathbf{A}}^2_{\text{sym}}\mathbf{X})$ and $g(\mathbf{A}^2_{\text{row}}\mathbf{X})$ in homophily and heterophily graphs, respectively, perfectly fitting our design choices. For the cases that the features themselves are already good for the classification, *SlenderGNN* gives both the node and propagated features large weights to achieve higher accuracy.

## 7 Conclusion

The main contribution of this work is *SlenderGNN*, which has a long list of desirable properties:

- *Accurate*: On both real-world and synthetic datasets, *SlenderGNN* exceeds the accuracy of state-of-the-art linear GNNs and it matches or exceeds the accuracy of nonlinear GNNs with $8\times - 64\times$ fewer parameters.
- *Robust*: *SlenderGNN* can handle the case of homophily, heterophily, and no-network-effect (randomly connected) graphs, as well as the case of no meaningful features.
- *Slender*: *SlenderGNN* has few, but useful parameters.
- *Interpretable*: *SlenderGNN* can easily justify its decisions (see Figure 1c).

Additional contributions include:

- The GNNLin framework, that illuminates the fundamental similarities and differences of popular GNN variants (see Table 1).
- The 'sanity checks', which highlight the strengths and weaknesses of each GNN method.

**Reproducibility**: Our source code and 'sanity checks' are available at https://bit.ly/3fhWJfK.

## REFERENCES


Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *ICLR*, 2021.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286 (5439):509–512, 1999.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *ICLR*, 2022.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020.

Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *ICLR*, 2021.

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

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2): 217–288, 2011.

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

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks. In *ICLR*, 2021.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *ICLR*, 2021.

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

Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*, 2019a.

Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019b.

Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, 2010.

Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

Mingjie Li, Xiaojun Guo, Yifei Wang, Yisen Wang, and Zhouchen Lin. $G^2$cn: Graph gaussian convolution networks with concentrated graph filters. In *ICML*, 2022.

Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *KDD*, 2020.

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

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *J. Complex Networks*, 9(2), 2021.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Mag.*, 29(3):93–106, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018.

Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *KDD*, 2009.

Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

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

Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. Dissecting the diffusion process in linear graph convolutional networks. *NeurIPS*, 2021.

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

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32 (1):4–24, 2021.

Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018.

Jaemin Yoo, Hyunsik Jeon, and U Kang. Belief propagation network for hard inductive semi-supervised learning. In *IJCAI*, 2019.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

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

# A LINEARIZATION PROCESSES

We present detailed processes to linearize various graph neural networks (GNN). Refer to Section 2 of the main paper for frequently used symbols. Linearization is done in two steps. First, we replace all activation functions with the identity function from the original definition of each GNN. Second, if the resulting layer function contains learnable parameters, we devise a replacement that is at least as expressive as the given function but containing no learnable parameters. This is because our goal of linearization is not just deriving a linear function with respect to $\mathbf{X}$, but understanding GNNs in relation to logistic regression based on our GNNLIN framework. We ignore the bias terms of linear layers for simplicity, without loss of generality.

## A.1 PROOF OF LEMMA 1: REPRESENTING LINEAR MODELS WITH GNNLIN

We prove Lemma 1 by representing each linear GNN with GNNLIN. Let $K \geq 0$ be a hyperparameter that determines the number of propagation steps in every GNN.

### A.1.1 SGC (WU ET AL., 2019)

SGC directly fits the definition of linearization with the following propagator function:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \tilde{\mathbf{A}}_{\text{sym}}^{K}\mathbf{X}. \tag{4}$$

### A.1.2 DGC (WANG ET AL., 2021)

DGC has two variants, DGC-Euler and DGC-DK, which have different propagator functions. We focus on DGC-Euler, which is used as the main model in their experiments. DGC is similar to SGC, except that it controls the strength of self-loops as follows:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = [(1 - T/K)\mathbf{I} + (T/K)\tilde{\mathbf{A}}_{\text{sym}}]^{K}\mathbf{X}, \tag{5}$$

where $T > 0$ is a hyperparameter. The self-loops become stronger if $T$ is closer to 0.

### A.1.3 $\text{S}^2$GC (ZHU & KONIUSZ, 2021)

$\text{S}^2$GC computes the summation of features propagated with different numbers of steps:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \sum_{k=1}^{K}(\alpha\mathbf{I} + (1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}^{k})\mathbf{X}. \tag{6}$$

The original formulation divides the added features by $K$, which can be safely ignored considering that the weight matrix $\mathbf{W}$ is multiplied to the transformed feature for classification.

### A.1.4 $\text{G}^2$CN (LI ET AL., 2022)

Since $\text{G}^2$CN does not provide an explicit formulation of the propagator function, we need to derive it. First, the parameterized version $\mathcal{P}'$ of the propagator function is given as follows:

$$\mathcal{P}'(\mathbf{A}, \mathbf{X}; \{\theta_i\}_{i=1}^{N}) = \sum_{i=1}^{N}\theta_i\mathbf{H}_{i,K}, \tag{7}$$

where $\theta_i$ is a learnable parameter. The $k$-th feature representation $\mathbf{H}_{i,k}$ is recursively defined as

$$\begin{aligned}
\mathbf{H}_{i,k} &= \mathbf{H}_{i,k-1} - \frac{T_i}{K}(\mathbf{L} - b_i\mathbf{I})^2\mathbf{H}_{i,k-1} \\
&= \mathbf{H}_{i,k-1} - \frac{T_i}{K}((b_i - 1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2\mathbf{H}_{i,k-1} \\
&= [\mathbf{I} - \frac{T_i}{K}((b_i - 1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]\mathbf{H}_{i,k-1} \\
\mathbf{H}_{i,0} &= \mathbf{X},
\end{aligned} \tag{8}$$

where $N$, $T_i$, and $b_i$ are hyperparameters, and $\mathbf{L} = \mathbf{I} - \mathbf{A}_{\text{sym}}$ is the normalized Laplacian matrix. Since the transformed features are combined with a learnable parameter $\theta_i$ in Equation 7, we make a propagator function $\mathcal{P}$ that contains no learnable parameters as follows:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \Big\|_{i=1}^{N} \mathbf{H}_{i,K} = \Big\|_{i=1}^{N} [\mathbf{I} - \frac{T_i}{K}((b_i - 1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]^K \mathbf{X}. \tag{9}$$

### A.2 Linearization of Decoupled GNNs

The propagation in decoupled GNNs is done on the abstract representations of node features, which are typically generated by multilayer perceptrons (MLP). Linearization starts with replacing MLPs with linear projections, and removes additional nonlinearity in the process of propagation.

#### A.2.1 PPNP (Klicpera et al., 2019a)

The lineraization of PPNP is straightforward, since the authors present a closed-form representation of the propagator function. If we remove the activation function, we have the following:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = (\mathbf{I}_n - (1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}})^{-1}\mathbf{X}, \tag{10}$$

where $0 < \alpha < 1$ is a hyperparameter that controls the weight of self-loops.

#### A.2.2 APPNP (Klicpera et al., 2019a)

We assume that the initial node representation is created by a single linear layer of $\mathbf{XW}$, where $\mathbf{W}$ is a weight matrix. Then, the $k$-th representation matrix $\mathbf{H}_k$ is represented as follows:

$$\mathbf{H}_k = (1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H}_{k-1} + \alpha\mathbf{XW}, \tag{11}$$

where $0 < \alpha < 1$ is a hyperparameter. The closed-form representation of $\mathbf{H}_K$ is given as follows:

$$\mathbf{H}_K = \left( [(1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}]^K + \alpha \sum_{k=0}^{K-1} [(1 - \alpha)\tilde{\mathbf{A}}_{\text{sym}}]^k \right) \mathbf{XW}. \tag{12}$$

We safely remove the weight matrix $\mathbf{W}$, which is redundant, and get the fiinal representation:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = [\sum_{k=0}^{K-1} \alpha(1 - \alpha)^k \tilde{\mathbf{A}}_{\text{sym}}^k + (1 - \alpha)^K \tilde{\mathbf{A}}_{\text{sym}}^K]\mathbf{X} \tag{13}$$

#### A.2.3 GDC (Klicpera et al., 2019b)

GDC generalizes APPNP and presents various forms of the propagation function. We pick the most representative one given in the paper, which is directly related to APPNP. The unnormalized version of the propagation matrix $\mathbf{S}'$ is given as follows:

$$\mathbf{S}' = \sum_{k=0}^{\infty} \alpha(1 - \alpha)^k \tilde{\mathbf{A}}_{\text{sym}}^k, \tag{14}$$

and then it is normalized and sparsified as

$$\mathbf{S} = \text{sparsify}(\tilde{\mathbf{S}}'_{\text{sym}}). \tag{15}$$

$\tilde{\mathbf{S}}'_{\text{sym}}$ represents adding self-loops and applying the symmetric normalization to $\mathbf{S}'$. The paper gives two approaches for sparsification, which are a) removing elements smaller than $\epsilon$, which is given as a hyperparameter, and b) selecting the top $k$ neighbors for each node. With any choice, the function $\mathcal{P}$ is simply given as follows:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \mathbf{SX}. \tag{16}$$

#### A.2.4 GPR-GNN (CHIEN ET AL., 2021)

We assume that the initial node representation is created by a single linear layer of $\mathbf{XW}$, where $\mathbf{W}$ is a weight matrix. Then, we have the following propagator function $\mathcal{P}'$ with parameters:

$$\mathcal{P}'(\mathbf{A}, \mathbf{X}; \{\theta_k\}_{k=0}^{K}) = \sum_{k=0}^{K} \theta_k \tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X}, \tag{17}$$

where $\theta_k$ is a parameter that is learned together with $\mathbf{W}$. We replace the summation with concatenation to remove the learnable parameters from $\mathcal{P}'$ and get the following:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \mathop{\Big\|}_{k=0}^{K} \tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X}. \tag{18}$$

### A.3 LINEARIZATION OF COUPLED GNNS

#### A.3.1 CHEBNET (DEFFERRARD ET AL., 2016)

Let $\mathbf{H}_k$ be the $k$-th node representation matrix, and $\mathbf{L} = \mathbf{I} - \mathbf{A}_{\text{sym}}$ be the graph Laplacian matrix normalized symmetrically. Then, the propagator function of ChebNet with parameters $\theta$ is

$$\mathcal{P}'(\mathbf{A}, \mathbf{X}; \theta) = \sum_{k=0}^{K-1} \theta_k \mathbf{H}_k, \tag{19}$$

where the recurrence relation is given as follows with the intial terms:

$$\begin{aligned} \mathbf{H}_0 &= \mathbf{X} \\ \mathbf{H}_1 &= \mathbf{LX} = -\mathbf{A}_{\text{sym}}\mathbf{X} + \mathbf{X} \\ &\cdots \\ \mathbf{H}_k &= 2(\mathbf{L} - \mathbf{I})\mathbf{H}_{k-1} - \mathbf{H}_{k-2} \\ &= -2\mathbf{A}_{\text{sym}}\mathbf{H}_{k-1} - \mathbf{H}_{k-2}. \end{aligned} \tag{20}$$

Based on the recurrence relation, the closed-form representation of $\mathbf{H}_k$ is given as

$$\mathbf{H}_k = a_k \mathbf{A}_{\text{sym}}^k \mathbf{X} + a_{k-1} \mathbf{A}_{\text{sym}}^{k-1} \mathbf{X} + \cdots + a_0 \mathbf{X}, \tag{21}$$

where $a_0, \cdots, a_k$ are constants. Since we have $K$ free parameters $\theta_0, \cdots, \theta_K$ corresponding to the $K$ terms in the representation matrix $\mathbf{H}_K$, we safely rewrite the propagator function as

$$\mathcal{P}'(\mathbf{A}, \mathbf{X}; \theta) = \sum_{k=0}^{K-1} \theta_k \mathbf{A}_{\text{sym}}^k \mathbf{X}. \tag{22}$$

Each value of $k$ has a free parameter $\theta_k$. Thus, we generalize it as follows:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \mathop{\Big\|}_{k=0}^{K-1} \mathbf{A}_{\text{sym}}^k \mathbf{X}. \tag{23}$$

#### A.3.2 GRAPHSAGE (HAMILTON ET AL., 2017)

We assume the mean aggregator of GraphSAGE. By replacing the activation function as the identity function, each layer $\mathcal{F}$ of GraphSAGE is linearized as follows:

$$\mathcal{F}(\mathbf{X}) = \mathbf{XW}_1 + \mathbf{A}_{\text{row}}\mathbf{XW}_2, \tag{24}$$

where $\mathbf{A}_{\text{row}} = \mathbf{D}^{-1}\mathbf{A}$ represents the mean operator in the aggregation, and $\mathbf{W}_1$ and $\mathbf{W}_2$ are learnable weight matrices in the layer. If we apply a chain of two layers, where the weight matrices of the second layer are represented as $\mathbf{W}_3$ and $\mathbf{W}_4$, we get the following:

$$\mathcal{F}(\mathcal{F}(\mathbf{X})) = \mathbf{XW}_1\mathbf{W}_3 + \mathbf{A}_{\text{row}}\mathbf{X}(\mathbf{W}_1\mathbf{W}_2 + \mathbf{W}_2\mathbf{W}_3) + \mathbf{A}_{\text{row}}^2\mathbf{XW}_2\mathbf{W}_4 \tag{25}$$

$$= \mathbf{XW}_a + \mathbf{A}_{\text{row}}\mathbf{XW}_b + \mathbf{A}_{\text{row}}^2\mathbf{XW}_c, \tag{26}$$

where we redefine the weight matrices without loss of generality as

$$\mathbf{W}_a = \mathbf{W}_1\mathbf{W}_3 \tag{27}$$
$$\mathbf{W}_b = \mathbf{W}_1\mathbf{W}_2 + \mathbf{W}_2\mathbf{W}_3 \tag{28}$$
$$\mathbf{W}_c = \mathbf{W}_2\mathbf{W}_4. \tag{29}$$

If we generalize it into $K$ layers, we get the following:

$$\mathcal{F}^K(\mathbf{X}) = \sum_{k=1}^{K} \mathbf{A}_{\text{row}}^k \mathbf{X}\mathbf{W}_k. \tag{30}$$

Note that a different weight matrix $\mathbf{W}_k$ is applied to each layer $k$. This is equivalent to concatenating the transformed features of all layers and learning a single large weight matrix in training.

#### A.3.3 GCNII (Chen et al., 2020)

After replacing the activation function with the identity function, the $l$-th layer $\mathcal{F}_l$ of GCNII is given as follows:

$$\mathcal{F}_l(\mathbf{H}) = ((1-\alpha_l)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha_l\mathbf{X}))((1-\beta_l)\mathbf{I} + \beta_l\mathbf{W}_l), \tag{31}$$

where $\alpha_l$ and $\beta_l$ are hyperparameters, and $\mathbf{W}_l$ is a weight matrix. The second term is equivalent to $\mathbf{W}_l$ regardless of the value of $\beta_l$, since $\mathbf{W}_l$ is a free parameter. We also set $\alpha_l$ to a constant $\alpha$ which is the same for every layer $l$, following the original paper (Chen et al., 2020).[1] Then, the equation is simplified as

$$\mathcal{F}_l(\mathbf{H}) = ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha\mathbf{X})\mathbf{W}_l. \tag{32}$$

If we apply a chain of two layers $l$ and $l+1$, we get the following:

$$\mathcal{F}_{l+1}(\mathcal{F}_l(\mathbf{H})) = ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha\mathbf{X})\mathbf{W}_l + \alpha\mathbf{X})\mathbf{W}_{l+1} \tag{33}$$
$$= (1-\alpha)^2\tilde{\mathbf{A}}_{\text{sym}}^2\mathbf{H}\mathbf{W}_l\mathbf{W}_{l+1} + \alpha(1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}\mathbf{W}_l\mathbf{W}_{l+1} + \alpha\mathbf{X}\mathbf{W}_{l+1} \tag{34}$$
$$= (1-\alpha)^2\tilde{\mathbf{A}}_{\text{sym}}^2\mathbf{H}\mathbf{W}_l' + \alpha(1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}\mathbf{W}_l' + \alpha\mathbf{X}\mathbf{W}_{l+1}, \tag{35}$$

where $\mathbf{W}_l' = \mathbf{W}_l\mathbf{W}_{l+1}$, which is also a free parameter. If we generalize it into $K$ layers, we get the following:

$$\mathcal{F}^K(\mathbf{X}) = (1-\alpha)^{K-1}((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^K + \alpha\tilde{\mathbf{A}}_{\text{sym}}^{K-1})\mathbf{X}\mathbf{W}_K + \alpha\sum_{k=0}^{K-2}(1-\alpha)^k\tilde{\mathbf{A}}_{\text{sym}}^k\mathbf{X}\mathbf{W}_k. \tag{36}$$

We safely remove the constant from each term, which can be included in the weight matrix:

$$\mathcal{F}^K(\mathbf{X}) = ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^K + \alpha\tilde{\mathbf{A}}_{\text{sym}}^{K-1})\mathbf{X}\mathbf{W}_K + \sum_{k=0}^{K-2}\tilde{\mathbf{A}}_{\text{sym}}^k\mathbf{X}\mathbf{W}_k. \tag{37}$$

We replace the summation operators between terms having different weight matrices with concatenation operators, having the final propagator function $\mathcal{P}$ as follows:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \Big\|_{k=0}^{K-2} \tilde{\mathbf{A}}_{\text{sym}}^k\mathbf{X} \,\|\, ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^K + \alpha\tilde{\mathbf{A}}_{\text{sym}}^{K-1})\mathbf{X} \tag{38}$$

### A.4 Linearization of Attention GNNs

#### A.4.1 DA-GNN (Liu et al., 2020)

We assume that the initial node representation is created by a single linear layer of $\mathbf{X}\mathbf{W}$, where $\mathbf{W}$ is a weight matrix. Then, the $k$-th representation matrix $\mathbf{H}_k$ is represented as follows:

$$\mathbf{H}_k = \tilde{\mathbf{A}}_{\text{sym}}^k\mathbf{X}\mathbf{W}. \tag{39}$$

[1] $\alpha$ is set to 0.1 in the original paper of GCNII (Chen et al., 2020).

DA-GNN computes the weighted sum of representations for all $k \in [0, K]$, where the weight values are determined also from the representation matrices:

$$\begin{aligned}\mathcal{P}(\mathbf{A}, \mathbf{X}) &= \sum_{k=0}^{K} \mathrm{diag}(\mathbf{H}_k \mathbf{s}) \mathbf{H}_k \\ &= \sum_{k=0}^{K} \mathrm{diag}(\tilde{\mathbf{A}}_{\mathrm{sym}}^k \mathbf{X}\mathbf{W}\mathbf{s}) \tilde{\mathbf{A}}_{\mathrm{sym}}^k \mathbf{X}\mathbf{W}.\end{aligned} \tag{40}$$

where $\mathbf{s}$ is a learnable weight vector. We safely remove the last $\mathbf{W}$ and rewrite $\mathbf{W}\mathbf{s}$ as $\mathbf{w}$. Then, we have the final representation of the propagator function:

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \sum_{k=0}^{K} \mathrm{diag}(\tilde{\mathbf{A}}_{\mathrm{sym}}^k \mathbf{X}\mathbf{w}) \tilde{\mathbf{A}}_{\mathrm{sym}}^k \mathbf{X}. \tag{41}$$

#### A.4.2 GAT (Velickovic et al., 2018)

We apply the following changes to linearize GAT, whose linearization is not straightforward due to the nonlinearity in the attention function:

1. We replace the activation functions between layers with the identity functions.
2. We simplify the attention function $\alpha_{ij} = \exp(e_{ij}) / \sum_k \exp(e_{ik})$ as $e_{ij}$.
3. We remove the LeaklyReLU function in the computation of $e_{ij}$.
4. We assume the single-head attention.

The edge weight $e_{ij}$, which is the $(i, j)$-th element of the propagator matrix, is defined as follows:

$$e_{ij} = \mathbf{a}_{\mathrm{dst}}^\top (\mathbf{W}^\top \mathbf{x}_i) + \mathbf{a}_{\mathrm{src}}^\top (\mathbf{W}^\top \mathbf{x}_j), \tag{42}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are feature vectors of length $d$ for node $i$ and $j$, respectively, $\mathbf{W}$ is a $d \times c$ learnable weight matrix, and $\mathbf{a}_{\mathrm{dst}}$ and $\mathbf{a}_{\mathrm{src}}$ are learnable weight vectors of length $c$. Then, we derive the initial form of a linearized GAT layer as follows:

$$\mathbf{H} = [\mathrm{diag}(\mathbf{X}\mathbf{W}\mathbf{a}_{\mathrm{dst}}) \tilde{\mathbf{A}} + \tilde{\mathbf{A}} \mathrm{diag}(\mathbf{X}\mathbf{W}\mathbf{a}_{\mathrm{src}})] \mathbf{X}\mathbf{W}. \tag{43}$$

Since all $\mathbf{a}_{\mathrm{dst}}$, $\mathbf{a}_{\mathrm{src}}$, and $\mathbf{W}$ are free parameters, we generalize it as follows:

$$\mathbf{H} = [\mathrm{diag}(\mathbf{X}\mathbf{w}_{\mathrm{dst}}) \tilde{\mathbf{A}} + \tilde{\mathbf{A}} \mathrm{diag}(\mathbf{X}\mathbf{w}_{\mathrm{src}})] \mathbf{X}\mathbf{W}, \tag{44}$$

where $\mathbf{w}_{\mathrm{dst}}$ and $\mathbf{w}_{\mathrm{src}}$ are learnable vectors of length $m$ that replace $\mathbf{a}_{\mathrm{dst}}$ and $\mathbf{a}_{\mathrm{src}}$, respectively.

## B Details on Sanity Checks

### B.1 Formal Definitions of Graph Scenarios

We categorize all possible scenarios of node classification based on the characteristics of node features $\mathbf{X}$, a graph structure $\mathbf{A}$, and node labels $\mathbf{y}$. We denote by $A_{ij}$ and $Y_i$ the random variables for edge $(i, j)$ between nodes $i$ and $j$ and label $y_i$ of node $i$, respectively.

**Edges** For the adjacency matrix $\mathbf{A}$, we consider the following three cases:

- **Uniform:** $P(A_{ki} = 1 \mid A_{ij} = A_{jk} = 1) = P(A_{ki} = 1)$
- **Clustered:** $P(A_{ki} = 1 \mid A_{ij} = A_{jk} = 1) > P(A_{ki} = 1)$
- **Bipartite:** $P(A_{ki} = 1 \mid A_{ij} = A_{jk} = 1) < P(A_{ki} = 1)$

The uniform case represents that every edge is determined independently of the others, and the graph structure gives no useful information for node classification. In the clustered case, it is more likely that nodes having common neighbors make more connections in a graph. In the bipartite case, nodes make more connections with those sharing no common neighbors.

**Labels** For the labels in $\mathbf{y}$, we consider the following three cases:

- **Individual:** $P(Y_i = y \mid A_{ij} = 1, Y_j = y) = P(Y_i = y)$
- **Homophily:** $P(Y_i = y \mid A_{ij} = 1, Y_j = y) > P(Y_i = y)$
- **Heterophily:** $P(Y_i = y \mid A_{ij} = 1, Y_j = y) < P(Y_i = y)$

The individual case means that the label of a node is independent of the labels of its neighbors. This is the case when the graph structure works as noise information with regard to classification. In the homophily case, adjacent nodes are likely to have the same label. It is the most popular assumption of GNNs for node classification. In the heterophily case, adjacent nodes are likely to have different labels, which is not as common as homophily but often observed in real-world graphs.

**Features** For the feature matrix $\mathbf{X}$, we consider the following three cases based on $\mathbf{A}$ and $\mathbf{y}$. We use the notation of $p(\cdot)$ since the features are typically modeled as continuous variables.

- **Random:** $p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j, a_{ij}) = p(\mathbf{x}_i, \mathbf{x}_j)$
- **Structural:** $p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j, a_{ij}) \neq p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j)$
- **Semantic:** $p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j, a_{ij}) \neq p(\mathbf{x}_i, \mathbf{x}_j \mid a_{ij})$

The random case represents that features are determined independently of the graph and labels. In this case, node features do not give useful information for classification, but work as unique indices of nodes like one-hot embeddings if the dimensionality $d$ of features is large. In the structural case, features are correlated with the graph structure, but not directly with labels. Such features give useful information for classification only if the graph structure and labels are related. In the semantic case, features directly provide useful information for predicting labels. Note that features can be semantic and structural at the same time by satisfying both of the conditions.

### B.2 Implementation of Graph Scenarios

There are various ways to generate synthetic graphs satisfying the definitions of different scenarios. One can use a synthetic graph generator designed to create more plausible graphs (Leskovec et al., 2010; Barabási & Albert, 1999), but we choose the simplest one to focus on the mutual information, rather than the other characteristics of synthetic graphs such as the degree distribution.

**Structure** We assume that the number of structural clusters is the same as the number $c$ of labels for the alignment with label information. We divide all nodes into $c$ groups and then decide the edge densities for intra- and inter-connections of groups based on the structural type: uniform, clustered, and bipartite. We use a hyperparameter $\epsilon_a$ to determine the noise level in the case of homophily or heterophily: for example, if $\epsilon_a = 0$, the graph has a full block-diagonal adjacency matrix in the case of homophily. The expected number of edges is the same for all three cases, and we set $\epsilon_a = 0$ in all of our experiments.

A notable characteristic of a bipartite structure $\mathbf{A}$ is that $\mathbf{A}^2$ has a clustered structure. If we create inter-group connections for every possible pair of different groups, even noise-free $\mathbf{A}$ with $\epsilon_a = 0$ makes noisy $\mathbf{A}^2$ with inter-group connections. For better consistency, we set the number of classes to an even number in our experiments, randomly pick paired classes such as $(1, 3)$ and $(2, 4)$ when $c = 4$, for example, and create inter-group connections only for the chosen pairs. In this way, we create a non-diagonal block-permutation matrix $\mathbf{A}$ when $\epsilon_a = 0$, and the noise level of $\mathbf{A}^2$, which has a clustered structure, is solely controlled by the noise level of $\mathbf{A}$.

**Labels** In the case of individual $\mathbf{y}$, we determine the label of each node uniformly at random, with no consideration of the graph structure. In the case of homophily or heterophily $\mathbf{y}$, we assign labels based on the groups of nodes assumed by the graph structure. That is, nodes in the same group have the same label, and thus the group index itself works as the label $\mathbf{y}$. In this way, we force homophily $\mathbf{y}$ for clustered $\mathbf{A}$, and heterophily $\mathbf{y}$ for bipartite $\mathbf{A}$. The degree of homophily (or heterophily) is also determined by the noise level $\epsilon_a$ of the graph structure.

**Features** We basically assume that every feature element is sampled from a uniform distribution. Thus, in the random case, we sample each element from the uniform distribution $\mathcal{U}(0, 1)$ between 0 and 1. In the structural case, we run low-rank support vector decomposition (SVD) (Halko et al., 2011) to make $\mathbf{X}$ have structural information. Given $\mathbf{U}\Sigma\mathbf{V}^\top \approx \mathbf{A}$ from low-rank SVD, we take $\mathbf{U}$ and normalize each feature element to have the zero-mean and unit-variance. The rank $r$ in the SVD is determined as a hyperparameter; higher $r$ captures the structure better, but can give noisy information. We also apply the ReLU function to $\mathbf{U}$ to make them positive.

Table 5: **Search spaces of hyperparameters.**

| **Method** | **Hyperparameters** |
|---|---|
| LR | $wd = [0, 5e^{-4}]$ |
| SGC | $wd = [0, 5e^{-4}], K = 2$ |
| DGC | $wd = [0, 5e^{-4}], K = 200, T = [3, 4, 5, 6]$ |
| $S^2$GC | $wd = [0, 5e^{-4}], K = 16, \alpha = [0.01, 0.03, 0.05, 0.07, 0.09]$ |
| $G^2$CN | $wd = [0, 5e^{-4}], K = 100, N = 2, T_1 = T_2 = [10, 20, 30, 40], b_1 = 0, b_2 = 2$ |
| GCN | $wd = [0, 5e^{-4}], lr = [2e^{-3}, 0.01, 0.05], K = 2$ |
| SAGE | $wd = [0, 5e^{-4}], lr = [2e^{-3}, 0.01, 0.05], K = 2$ |
| GCNII | $wd = [0, 5e^{-4}], lr = 0.01, K = [8, 16, 32, 64], \alpha = [0.1, 0.2, 0.5], \theta = [0.5, 1, 1.5]$ |
| APPNP | $wd = [0, 5e^{-4}], lr = [2e^{-3}, 0.01, 0.05], K = 10, \alpha = 0.1$ |
| GPR-GNN | $wd = [0, 5e^{-4}], lr = [2e^{-3}, 0.01, 0.05], K = 10, \alpha = [0.1, 0.2, 0.5, 0.9]$ |
| GAT | $wd = [0, 5e^{-4}], lr = [2e^{-3}, 0.01, 0.05], K = 2, heads = 8$ |
| *SlenderGNN* | $wd_1 = [1e^{-3}, 1e^{-4}, 1e^{-5}], wd_2 = [1e^{-3}, 1e^{-4}, 1e^{-5}, 1e^{-6}]$ |

Table 6: ***SlenderGNN* is slender (i.e., contains only a few parameters).** We report the number of thousand parameters on real-world graphs for all competing models.

| **Model** | **Cora** | **CiteSeer** | **PubMed** | **Computers** | **Photo** | **Chameleon** | **Squirrel** | **Actor** | **Penn94** |
|---|---|---|---|---|---|---|---|---|---|
| LR | 10.0 | 22.2 | 1.5 | 7.7 | 6.0 | 11.6 | 10.5 | 4.7 | 9.6 |
| SGC | 10.0 | 22.2 | 1.5 | 7.7 | 6.0 | 11.6 | 10.5 | 4.7 | 9.6 |
| DGC | 10.0 | 22.2 | 1.5 | 7.7 | 6.0 | 11.6 | 10.5 | 4.7 | 9.6 |
| $S^2$GC | 10.0 | 22.2 | 1.5 | 7.7 | 6.0 | 11.6 | 10.5 | 4.7 | 9.6 |
| $G^2$CN | 20.1 | 44.4 | 3.0 | 15.4 | 11.9 | 23.3 | 20.9 | 9.3 | O.O.M. |
| GCN | 92.2 | 237 | 32.3 | 49.8 | 48.3 | 149 | 134 | 60.0 | 308 |
| SAGE | 184 | 475 | 64.5 | 99.5 | 96.5 | 298 | 268 | 120 | O.O.M. |
| GCNII | 1255 | 270 | 65.0 | 82.6 | 81.0 | 182 | 167 | 92.8 | 341 |
| APPNP | 92.2 | 237 | 32.3 | 49.8 | 48.3 | 149 | 134 | 60.0 | 308 |
| GPR-GNN | 92.2 | 237 | 32.3 | 49.8 | 48.3 | 149 | 134 | 60.0 | 308 |
| GAT | 739 | 190 | 25.9 | 399 | 387 | 1195 | 1074 | 481 | 2467 |
| *SlenderGNN* | 29.7 | 35.7 | 6.0 | 30.7 | 23.9 | 2.5 | 4.2 | 18.7 | 38.5 |

In the semantic case, we randomly pick $c$ representative vectors $\{\mathbf{v}_k\}_{k=1}^c$ from the uniform distribution, which correspond to the $c$ different classes. Then, for each node $i$ with label $y$, we sample a feature vector such that $\arg\max_k \mathbf{x}^\top \mathbf{v}_k = y$. In this way, we have random vectors having sufficient semantic information for the classification of labels, with a guarantee that the perfect linear decision boundaries can be drawn in the feature space $\mathbf{X}$ at the training time.

## C Reproducibility

We perform row-normalization on the node features of all datasets as done in most studies on GNNs. We report the hyperparameters used for a grid search in Table 5, which is done for every split of data. The dimensions of hidden layers are all set to $64$, and the probabilities of dropout layers are all set to $0.5$. For the linear models, we use L-BFGS as the optimizer for training $100$ epochs with patience $5$; for the nonlinear ones, we use ADAM and train them for $1000$ epochs with patience $200$.

In *SlenderGNN*, there are only 2 hyperparameters, where $wd_1$ is the weight of LASSO, and $wd_2$ is the weight of group LASSO. It is worth noting that, when searching the hyperparameters, *SlenderGNN* does not need to recompute the features, while most of the linear methods need to do so because of including one or more hyperparameters in the features that need to be precomputed.