Less is More: SLIMG for Accurate, Robust, and Interpretable Graph Mining

Jaemin Yoo*
Carnegie Mellon University
Pittsburgh, USA
jaeminyoo@cmu.edu

Meng-Chieh Lee*
Carnegie Mellon University
Pittsburgh, USA
mengchil@cs.cmu.edu

Shubhranshu Shekhar
Carnegie Mellon University
Pittsburgh, USA
shubhras@andrew.cmu.edu

Christos Faloutsos
Carnegie Mellon University
Pittsburgh, USA
christos@cs.cmu.edu

ABSTRACT

How can we solve semi-supervised node classification in various graphs possibly with noisy features and structures? Graph neural networks (GNNs) have succeeded in many graph mining tasks, but their generalizability to various graph scenarios is limited due to the difficulty of training, hyperparameter tuning, and the selection of a model itself. Einstein said that we should "make everything as simple as possible, but not simpler." We rephrase it into the *careful simplicity* principle: a carefully-designed simple model can surpass sophisticated ones in real-world graphs. Based on the principle, we propose SLIMG for semi-supervised node classification, which exhibits four desirable properties: It is (a) *accurate*, winning or tying on 10 out of 13 real-world datasets; (b) *robust*, being the only one that handles all scenarios of graph data (homophily, heterophily, random structure, noisy features, etc.); (c) *fast and scalable*, showing up to **18**× faster training in million-scale graphs; and (d) *interpretable*, thanks to the linearity and sparsity. We explain the success of SLIMG through a systematic study of the designs of existing GNNs, sanity checks, and comprehensive ablation studies.

CCS CONCEPTS

• **Computing methodologies** → Machine learning algorithms.

KEYWORDS

graph neural networks, semi-supervised node classification

ACM Reference Format:
Jaemin Yoo, Meng-Chieh Lee, Shubhranshu Shekhar, and Christos Faloutsos. 2023. Less is More: SLIMG for Accurate, Robust, and Interpretable Graph Mining. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599413

1 INTRODUCTION

How can we solve semi-supervised node classification in various types of graphs possibly with noisy features and structures? Graph neural networks (GNNs) [7, 9, 14, 31] have succeeded in various graph mining tasks such as node classification, clustering, or link prediction. However, due to the difficulty of training, hyperparameter tuning, and even the selection of a model itself, many GNNs fail to show their best performance when applied to a large testbed that contains real-world graphs with various characteristics. Especially when a graph contains noisy observations in its features and/or its graphical structure, which is common in real-world data, existing models easily overfit their parameters to such noises.

Graph	Feature	SGC	S^2GC	SAGE	GCNII	GPR	SlimG
Noisy	Semantic		✓	✓			✓
Homophily	Noisy						✓
Heterophily	Noisy						✓
Homophily	Structural	✓	✓	✓	✓	✓	✓
Heterophily	Structural	✓		✓		✓	✓
Homophily	Semantic	✓	✓	✓	✓	✓	✓
Heterophily	Semantic	✓		✓	✓	✓	✓

Figure 1: SLIMG wins on all sanity checks. Each row is a specific scenario of graph data that we propose for comprehensive evaluation in Section 5. The table is generated from the actual accuracy in Table 2: ✓ means the accuracy ≥ 80%.

In response to the question, we propose SLIMG, our novel classifier model on graphs based on the *careful simplicity* principle: a simple carefully-designed model can be more accurate than complex ones thanks to better generalizability, robustness, and easier training. The four design decisions of SLIMG (D1-4 in Sec. 4) are carefully made to follow this principle by observing and addressing the pain points of existing GNNs; we generate and combine various types of graph-based features (D1), design structure-only features (D2), remove redundancy in feature transformation (D3), and make the propagator function contain no hyperparameters (D4).

The resulting model, SLIMG, is our main contribution (C1) which exhibits the following desirable properties:

- **C1.1 - Accurate** on both real-world and synthetic datasets, almost always winning or tying in the first place (see Figure 2, Table 2, and Table 3).
- **C1.2 - Robust**, being able to handle numerous real settings such as homophily, heterophily, no network effects, useless features (see Figure 1 and Table 2).
- **C1.3 - Fast and scalable**, using carefully chosen features, it takes only 32 seconds on million-scale real-world graphs (ogbn-Products) on a stock server (see Figure 2).
- **C1.4 - Interpretable**, learning the largest weights on informative features, ignoring noisy ones, based on the linear decision function (see Figure 5).

*Both authors contributed equally to this research.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599413

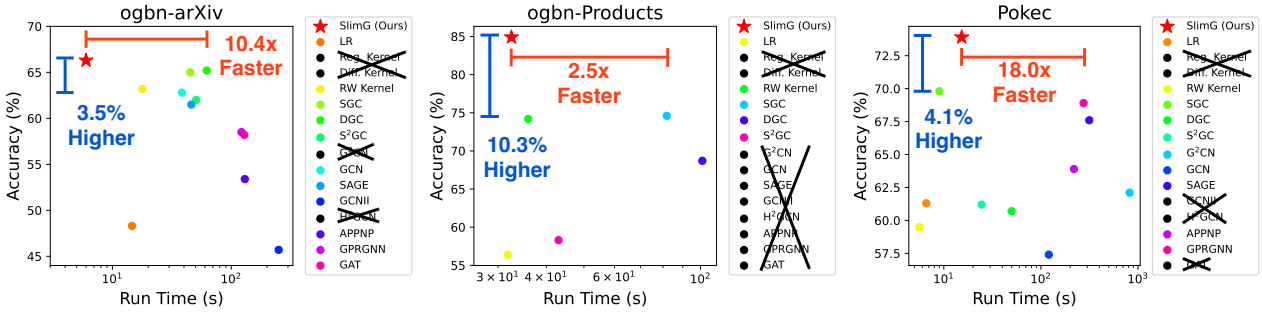


Figure 2: SlimG wins both on accuracy and training time on (left) ogbn-arXiv, (middle) ogbn-Products, and (right) Pokec, which are large real-world graphs (1.2M, 61.9M, and 30.6M edges, resp.). Several baselines run out of memory (crossed out).

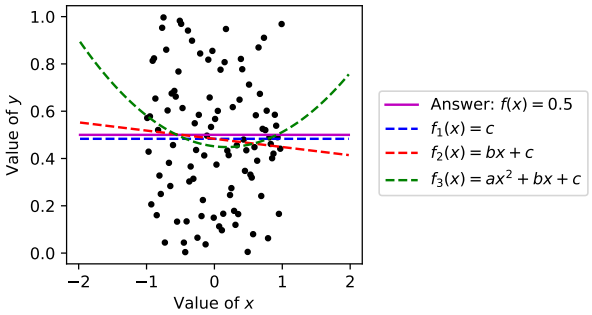


Figure 3: Why 'less is more': The simple model $f_0(x) = c$ (in blue) matches the reality (in solid purple), while richer models with more polynomial powers end up capturing noise in the given data: the tiny downward trend by $f_1(x)$ (in red) and the spurious curvature by $f_2(x)$ (in green).

Not only we propose a carefully designed, effective method (in Sec. 4), but we also explain the reasons for its success. This is thanks to our three additional contributions (C2-4):

- **C2 - Explanation (Sec. 3):** We propose GnnExp, a framework for the systematic linearization of a GNN. As shown in Table 1, GnnExp highlights the similarities, differences, and weaknesses of successful GNNs.
- **C3 - Sanity checks (Sec. 5):** We propose seven possible scenarios of graphs (homophily, heterophily, no network effects, etc.), which reveal the strong and weak points of each GNN; see Figure 1 with more details in Table 2.
- **C4 - Experiments (Sec. 6):** We conduct extensive experiments to better understand the success of SlimG even with its simplicity. Our results in Tables 5 to 9 show that SlimG effectively selects the most informative component in each dataset, fully exploiting its robustness and generality.

Less is more. Our justification for the counter-intuitive success of simplicity is illustrated in Figure 3: A set of points are uniformly distributed in $x \in (-1, 1)$ and $y \in (0, 1)$, and the fitting polynomials $f_i(x)$ with degree $i = 0$, 1 and 2 are given. Notice that the simplest model f_0 (blue line) matches the true generator $f(x) = 0.5$. Richer models use the 1st and the 2nd degree powers (*many cooks spoil the broth*) and end up modeling tiny artifacts, like the small downward slope of f_1 (red line), and the curvature of f_2 (green line). This 'many cooks' issue is more subtle and counter-intuitive than overfitting, as f_2 and f_3 have only 2 to 3 unknown parameters to fit to the hundred data points: even a *small* statistical can fail if it is not matched with the underlying data-generating mechanism.

Reproducibility. Our code, along with our datasets for *sanity checks*, is available at https://github.com/mengchillee/SlimG.

2 BACKGROUND AND RELATED WORKS

2.1 Background

We define 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** The labels $\mathbf{y} \in \{1, \cdots, c\}^m$ of m nodes in G, where $m \ll n$, and c is the number of classes.
- **Predict** The unknown classes of $n - m$ test nodes.

We use the following symbols to represent adjacency matrices with various normalizations and/or self-loops. $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-loops. $\tilde{\mathbf{D}} = \text{diag}(\tilde{\mathbf{A}}\mathbf{1}_{n\times 1})$ 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 the symmetrically normalized $\mathbf{A}$ but without self-loops. We also use a different type of normalization $\mathbf{A}_{\text{row}} = \mathbf{D}^{-1}\mathbf{A}$ (and accordingly $\tilde{\mathbf{A}}_{\text{row}}$), which we call row normalization, based on the position of the matrix $\mathbf{D}$.

As a background, we define logistic regression (LR) as a function to find the weight matrix $\mathbf{W}$ that best maps given features to labels with a linear function.

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 is the number of observations such that $m \le n$, 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) is a function that finds an optimal weight matrix $\mathbf{W} \in \mathbb{R}^{d\times c}$ from $\mathbf{X}$ and $\mathbf{y}$ as follows:*

$$\text{LR}(\mathbf{X}, \mathbf{y}) = \arg\max_{\mathbf{W}} \sum_{i=1}^{m} \sum_{j=1}^{c} y_{ij} \log \text{softmax}_j(\mathbf{W}^\top \mathbf{x}_i), \qquad (1)$$

where $\text{softmax}_j(\cdot)$ represents selecting the j-th element of the result of the softmax function. We omit the bias term for brevity.

2.2 Related Works

We introduce related works categorized into graph neural networks (GNN), linear GNNs, and graph kernel methods.

Graph neural networks. There exist many recent GNN variants; recent surveys [34, 38] group them into spectral models [5, 14], sampling-based models [9, 36, 40], attention-based models [2, 13, 31], and deep models with residual connections [3, 19]. Decoupled models [4, 15, 16] separate the two major functionalities of GNNs: the node-wise feature transformation and the propagation. GNNs are often fused with graphical inference [11, 37] to further improve the predicted results. These GNNs have shown great performance in many graph mining tasks, but suffer from limited robustness when applied to graphs with various characteristics possibly having noisy observations, especially in semi-supervised learning.

Linear graph neural networks. Wu et al. [33] proposed SGC by removing the nonlinear activation functions of GCN [14], reducing the propagator function to a simple matrix multiplication. Wang et al. [32] and Zhu and Koniusz [39] improved SGC by manually adjusting the strength of self-loops with hyperparameters, increasing the number of propagation steps. Li et al. [20] proposed G^{2}CN, which improves the accuracy of DGC [32] on heterophily graphs by combining multiple propagation settings (i.e. bandwidths). The main limitation of these models is the high complexity of propagator functions with many hyperparameters, which impairs both the robustness and interpretability of decisions even with linearity.

Graph kernel methods. Traditional works on graph kernel methods [12, 28] are closely related to linear GNNs, which can be understood as applying a linear graph kernel to transform the raw features. A notable limitation of such kernel methods is that they are not capable of addressing various scenarios of real-world graphs, such as heterophily graphs, as their motivation is to aggregate all information in the local neighborhood of each node, rather than ignoring noisy and useless ones. We implement three popular kernel methods as additional baselines and show that our SlimG outperforms them in both synthetic and real graphs.

3 PROPOSED FRAMEWORK: GNNEXP

Why do GNNs work well when they do? In what cases will a GNN fail? We answer these questions with GnnExp, our proposed framework for revealing the essence of each GNN. The idea is to derive the essential *feature propagator* function on which each variant is based, ignoring nonlinearity, so that all models are comparable on the same ground. The observations from GnnExp motivate us to propose our method SlimG, which we describe in Section 4.

Definition 2 (Linearization). *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 θ is the set of parameters. Then, f is* linearized *if $\theta = \{\mathbf{W}\}$ and the 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 without loss of generality.

Definition 3 (GnnExp). *Given a GNN f, GnnExp 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 $\mathcal{P}$.*

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

Table 1: GnnExp encompasses popular GNNs. The * and ** superscripts mark fully and partially linearized models, respectively. We derive Pain Points (Sec. 3.1) and Distinguishing Factors (Sec. 3.2) of the variants through GnnExp.

Model	Type	Propagator function $\mathcal{P}(\mathbf{A}, \mathbf{X})$
LR	Linear	$\mathbf{X}$
SGC	Linear	$\tilde{\mathbf{A}}_{\text{sym}}^K \mathbf{X}$
DGC	Linear	$[(1-T/K)\mathbf{I} + (T/K)\tilde{\mathbf{A}}_{\text{sym}}]^K \mathbf{X}$
S^{2}GC	Linear	$\sum_{k=1}^K (\alpha\mathbf{I} + (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^k)\mathbf{X}$
G^{2}CN	Linear	$\|_{i=1}^N [\mathbf{I} - (T_i/K)((b_i-1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]^K \mathbf{X}$
PPNP*	Decoupled	$(\mathbf{I} - (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}})^{-1}\mathbf{X}$
APPNP*	Decoupled	$[\sum_{k=0}^{K-1} \alpha(1-\alpha)^k \tilde{\mathbf{A}}_{\text{sym}}^k + (1-\alpha)^K \tilde{\mathbf{A}}_{\text{sym}}^K]\mathbf{X}$
GDC*	Decoupled	$\mathbf{S} = \text{sparse}_\epsilon(\sum_{k=0}^\infty (1-\alpha)^k \tilde{\mathbf{A}}_{\text{sym}}^k)$ for $\tilde{\mathbf{S}}_{\text{sym}}\mathbf{X}$
GPR-GNN*	Decoupled	$\|_{k=0}^K \tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X}$
ChebNet*	Coupled	$\|_{k=0}^{K-1} \mathbf{A}_{\text{sym}}^k \mathbf{X}$
GCN*	Coupled	$\tilde{\mathbf{A}}_{\text{sym}}^K \mathbf{X}$
SAGE*	Coupled	$\|_{k=0}^K \mathbf{A}_{\text{row}}^k \mathbf{X}$
GCNII*	Coupled	$\|_{k=0}^{K-2} \tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X} \,\|\, ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^K + \alpha\mathbf{A}_{\text{sym}}^{K-1})\mathbf{X}$
H$_2$GCN*	Coupled	$\|_{k=0}^{2K} \mathbf{A}_{\text{sym}}^k \mathbf{X}$
GAT**	Attention	$\prod_{k=1}^K [\text{diag}(\mathbf{X}\mathbf{w}_{k,1})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\text{diag}(\mathbf{X}\mathbf{w}_{k,2})]\mathbf{X}$
DA-GNN**	Attention	$\sum_{k=0}^K \text{diag}(\tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X}\mathbf{w})\tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X}$

Lemma 1. *GnnExp includes existing linear graph neural networks as its special cases: SGC, DGC, S^{2}GC, and G^{2}CN.*

Proof. The proof is given in Appendix A.1. ∎

In Table 1, we conduct a comprehensive linearization of existing GNNs using GnnExp to understand the fundamental similarities and differences among GNN variants. The models are categorized into linear, decoupled, coupled, and attention models. We ignore bias terms for simplicity, without loss of generality. Refer to Appendix B for details of the linearization process for each model.

3.1 Pain Points of Existing GNNs

Based on the comprehensive linearization in Table 1, we derive four pain points of existing GNNs which we address in Section 4.

Pain Point 1 (Lack of robustness). *All models in Table 1 fail to handle multiple graph scenarios at the same time, i.e., graphs with homophily, heterophily, no network effects, or useless features.*

Most models in Table 1 make an implicit assumption on given graphs, such as homophily or heterophily, rather than being able to perform well in multiple scenarios at the same time. For example, all models except ChebNet, SAGE, and H$_2$GCN have self-loops in the new adjacency matrix, emphasizing the local neighborhood of each node even in graphs with heterophily or no network effects. This is the pain point that we also observe empirically from the sanity checks (in Table 2), where none of the existing models succeeds in making reasonable accuracy in all cases of synthetic graphs.

Pain Point 2 (Vulnerability to noisy features). *All models in Table 1 cannot fully exploit the graph structure if the features are noisy, since they depend on the node feature matrix $\mathbf{X}$.*

If a graph does not have a feature matrix $\mathbf{X}$, a common solution is to introduce one-hot features [14], i.e., $\mathbf{X} = \mathbf{I}$, although it increases the running time of the model a lot. On the other hand, if $\mathbf{X}$ exists but some of its elements are meaningless with respect to the target classes, models whose propagator functions rely on $\mathbf{X}$ suffer from the noisy elements. In such cases, a desirable property for a model is to adaptively emphasize important features or disregard noisy ones to maximize its generalization performance, which is not satisfied by any of the existing models in Table 1.

Pain Point 3 (Efficiency and effectiveness)**.** *Concatenation-based models in Table 1 create spurious correlations between feature elements, requiring more parameters than in other models.*

Existing models such as GPR-GNN, SAGE, and GCNII in Table 1 perform the concatenation of multiple feature matrices transformed in different ways. For example, GPR-GNN concatenates $\tilde{\mathbf{A}}_{\text{sym}}^{k}\mathbf{X}$ for different values of k from 0 to K, where K is a hyperparameter. Such a concatenation-based propagation limits the efficiency of a model in two ways. First, this increases the number of parameters K times, since the model needs to learn a separate weight matrix for each given feature matrix. Second, this creates spurious correlations in the resulting features, since the feature matrices like $\tilde{\mathbf{A}}_{\text{sym}}\mathbf{X}$ and $\tilde{\mathbf{A}}_{\text{sym}}^{2}\mathbf{X}$ have high correlations with each other.

Pain Point 4 (Many hyperparameters)**.** *Hyperparameters in $\mathcal{P}$ impair its interpretability and require extensive tuning.*

Most models in Table 1, even the linear models such as DGC and G^2CN, contain many hyperparameters in the propagator function $\mathcal{P}$. Such hyperparameters lead to two limitations. First, the interpretability of the weight matrix $\mathbf{W}$ is impaired, since it is learned on top of the transformed feature $\mathcal{P}(\mathbf{A}, \mathbf{X})$ whose meaning changes arbitrarily by the choice of its hyperparameters. For example, DGC changes the number K of propagation steps between 250 and 900 in real-world datasets, making it hard to have consistent observations from the generated features. Second, $\mathcal{P}(\mathbf{A}, \mathbf{X})$ should be computed for every new choice of hyperparameters, while it can be cached and reused for searching hyperparameters outside $\mathcal{P}$.

3.2 Distinguishing Factors

What potential choices do we have in designing a general approach that addresses the pain points? We analyze the fundamental similarities and differences among the GNN variants in Table 1.

Distinguishing Factor 1 (Combination of features)**.** *How should we combine 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. There are two main choices in Table 1: **(1)** the summation of transformed features (most models), and **(2)** the concatenation of features (GPR-GNN, GraphSAGE, GCNII, and H_2GCN). Simple approaches like SGC are categorized as the summation due to the self-loops in $\tilde{\mathbf{A}}_{\text{sym}}$.

Distinguishing Factor 2 (Modification of $\mathbf{A}$)**.** *How should we normalize or modify the adjacency matrix $\mathbf{A}$?*

The three prevailing choices are given as follows: **(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 $\tilde{\mathbf{A}}_{\text{sym}}$ with self-loops, but some variants avoid self-loops and use either row normalization $\mathbf{A}_{\text{row}}$ or symmetric one $\mathbf{A}_{\text{sym}}$. Recent models such as DGC, G^2CN, and GCNII determine the weight of self-loops with hyperparameters, since strong self-loops allow distant propagation with a large value of K. Finally, attention-based models learn the elements in $\mathbf{A}$ dynamically based on node features, making propagator functions quadratic with $\mathbf{X}$, not linear.

Distinguishing Factor 3 (Heterophily)**.** *What to do if the direct neighbors differ in their features or labels?*

In such cases, the simple aggregation of the features of immediate neighbors may hurt performance, and therefore, several GNNs do suffer under heterophily as shown in Table 2. GNNs that can handle heterophily adopt one or more of these ideas: **(1)** using the square of $\mathbf{A}$ as the base structure (G^2CN); **(2)** learning different weights for different steps (GPR-GNN, ChebNet, SAGE, and GCNII), and **(3)** making small or no self-loops in the modification of $\mathbf{A}$ (DGC, S^2GC, G^2CN, and H^2GCN). The idea 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 the implicit summation of transformed features.

4 PROPOSED METHOD: SLIMG

We propose SlimG, a novel method that addresses the limitations of existing models with strict adherence to the *careful simplicity* principle. We first derive four *design decisions* (D1-D4) that directly address the pain points (PPs) of existing GNN models and propose the following propagator function of SlimG:

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

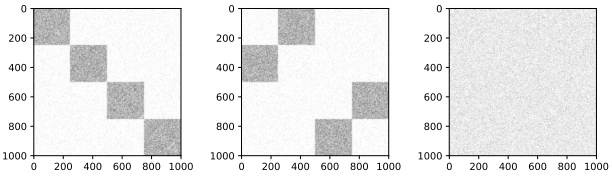
where $g(\cdot)$ is the principal component analysis (PCA) for the orthogonalization of each component, followed by an L2 normalization, and $\mathbf{U} \in \mathbb{R}^{n \times r}$ contains r-dimensional structural features independent of node features $\mathbf{X}$, derived by running the low-rank singular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

***D1**: Concatenating winning normalizations (for PP 1 - robustness).* The main principle of SlimG to acquire robustness and generalizability, in response to Pain Point 1, is to transform the raw features into various forms and then combine them through concatenation. In this way, SlimG is able to emphasize essential features or ignore useless ones by learning separate weights for different components. The four components of SlimG in Equation 3 show their strength in different cases: structural features $\mathbf{U}$ for graphs with noisy features, self-features $\mathbf{X}$ for a noisy structure, two-step aggregation $\mathbf{A}_{\text{row}}^{2}$ for heterophily graphs, and smoothed two-hop aggregation $\tilde{\mathbf{A}}_{\text{sym}}^{2}$ of the local neighborhood for homophily.

Specifically, we use the row-normalized matrix $\mathbf{A}_{\text{row}}$ with no self-loops due to the limitations of the symmetric normalization $\tilde{\mathbf{A}}_{\text{sym}}$: First, the self-loops force one to combine all intermediate neighbors of each node until the K-hop distance, even in heterophily graphs where the direct neighbors should be avoided. Second, neighboring

Structure	Feature		
	Semantic **X**	Structural **X**	Random **X**
Homophily **A**	Both help	Both help	**A** helps
Heterophily **A**	Both help	Both help	**A** helps
Uniform **A**	**X** helps	None helps	None helps

(a) Sanity check matrix



(b) Homophily A **(c) Heterophily A** **(d) Uniform A**

Figure 4: Illustration of our sanity checks. (a) We consider 3 possibilities for each of A and X, creating a total of 9 cases. (b - d) We visualize the adjacency matrices from the three cases of A, which exhibit different structural patterns.

features are rescaled based on the node degrees during an aggregation, even when we want simple aggregation of K-hop neighbors preserving the original scale of features. We thus use $\mathbf{A}^2_{\text{row}}$ along with the popular transformation $\tilde{\mathbf{A}}^2_{\text{sym}}$ in SLIMG.

***D2**: Structural features (for PP 2 - noisy features).* In response to Pain Point 2, we have to resort to the structure **A** ignoring **X** when features are missing, noisy, or useless for classification. Thanks to our design decision (D1) for concatenating different components, we can safely add to $\mathcal{P}$ structure-based features which the model can utilize adaptively based on the amount of information **X** provides. However, it is not effective to use raw **A**, which requires the model to learn a separate weight vector for each node, severely limiting its generalizability. We thus adopt low-rank SVD with rank r to extract structural features **U**. The value of r is selected to keep 90% of the energy of **A**, where the sum of the largest r squared singular values divided by the squared Frobenius norm of **A** is smaller than 0.9. If the chosen value of r is larger than d in large graphs, we set r to d for the size consistency between different components.

***D3**: Orthogonalization and sparsification (for PP 3 - collinearity).* To improve the efficiency and effectiveness of SLIMG in response to Pain Point 3, we use two reliable methods to further transform the features: dimensionality reduction by PCA and regularization by group LASSO. First, we run PCA on each of the four components to orthogonalize them and to improve the consistency of learned weights. Second, we use group LASSO to learn sparse weights on the component level, preserving the relative magnitude of each element and suppressing noisy features. To assure the consistency between components (especially with the structural features **U**), we force all components to have the same dimensionality by selecting r features from each component when adopting PCA.

***D4**: Multi-level neighborhood aggregation (for PP 4 - hyperparameters).* Our propagator function $\mathcal{P}$ considers multiple levels of neighborhoods through the concatenation of different components. This allows us to remove all hyperparameters from $\mathcal{P}$ to tune for each dataset, in response to Pain Point 4, gaining in both interpretability and efficiency. Specifically, **X**, $\tilde{\mathbf{A}}^2_{\text{sym}}$, and $\mathbf{A}^2_{\text{row}}$ aggregate the zero-, one-, and two-hop neighborhood of each node, respectively, considering the self-loops included in $\tilde{\mathbf{A}}^2_{\text{sym}}$. Then, **U** considers the global topology information of each node, which is in effect the same as considering the distant neighborhood in the graph. As a result, SLIMG performs well in different real-world datasets without tuning any hyperparameters in $\mathcal{P}$, unlike existing GNNs that are often required to increase the value of K up to hundreds [32].

Time complexity. The time complexity of SLIMG, including all its components SVD, PCA, and the training of LR, is linear with a graph size in most real-world graphs where the number of edges is much larger than the numbers of nodes and features.

Lemma 2. *Given a graph, let n and e be the numbers of nodes and edges, respectively, and d be the number of features. Then, the time complexity of the training of SLIMG is $O(de + d^2n + d^3)$.*

PROOF. The proof is given in Appendix A.2. ∎

5 PROPOSED SANITY CHECKS

We propose a set of *sanity checks* to directly evaluate the robustness of GNNs to various scenarios of node classification.

5.1 Design of Sanity Checks

We categorize possible scenarios of node classification based on the characteristics of node features **X**, a graph structure **A**, and node labels **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. We summarize the nine possible cases in Figure 4a.

Structure. We consider three cases of the structure **A**: *uniform*, *homophily*, and *heterophily*, which are defined as follows:

- **Uniform:** $P(Y_i = y \mid A_{ij} = 1, Y_j = y) = P(Y_i = y)$
- **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 uniform case means that the label of a node is independent of the labels of its neighbors. This is the case when the graph structure provides no information for classification. In the homophily case, adjacent nodes are likely to have the same label, which is the most common assumption in graph data. 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.

We assume the one-to-one correspondence between the structural property and the labels: *uniform* with uniformly random **A**, *homophily* with block-diagonal **A**, and *heterophily* with non-block-diagonal **A**. Note that the other combinations, e.g., homophily with non-block-diagonal **A**, or uniform with block-diagonal **A**, are not feasible by definition. Figure 4 illustrates the three cases of **A**. The number of node clusters in the graph, which is four in the figure, is the same as the number of node labels in experiments.

Features. We consider three cases of node features **X**: *random*, *semantic*, and *structural*. The three cases are defined in relation to **A** and **y**. We use the notation $p(\cdot)$ 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:** $p(\mathbf{x}_i, \mathbf{x}_j \mid y_i, y_j, a_{ij}) \neq p(\mathbf{x}_i, \mathbf{x}_j \mid a_{ij})$

Table 2: SLIMG wins on all sanity checks. Each value denotes the average and the standard deviation of accuracy from five runs. There are three groups of scenarios: (left) only features X help; (middle) only connectivity A helps; (right) both help. Green (, ,) marks the top three (higher is darker); red ( ) marks the ones that are too low (2σ below the third place). SLIMG is the only method without red cells and achieves the best average accuracy and average rank (variance in the parentheses).

Model	Only **X** helps	Only **A** helps		Both **X** and **A** help				Avg. Acc	Avg. Rank
	Semantic **X** Uniform **A**	Random **X** Homophily	Random **X** Heterophily	Structural **X** Homophily	Structural **X** Heterophily	Semantic **X** Homophily	Semantic **X** Heterophily		
LR	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	62.4 (26.9)	10.7 (5.4)
Reg. Kernel	82.7±0.5	27.9±0.4	24.3±1.0	75.7±0.2	65.3±1.6	91.5±0.5	79.5±0.3	63.8 (27.0)	10.4 (4.3)
Diff. Kernel	26.8±1.7	38.0±8.7	37.6±7.5	79.5±0.3	73.5±0.6	70.9±23.	56.1±27.	54.6 (20.7)	10.6 (4.0)
RW Kernel	72.2±0.7	37.0±0.4	24.5±1.3	81.3±1.2	51.0±1.1	94.5±0.9	57.8±0.7	59.8 (24.7)	10.4 (3.6)
SGC	44.6±9.8	64.3±0.7	50.2±14.	87.1±0.6	84.3±0.5	93.9±0.9	91.5±0.5	73.7 (20.4)	5.7 (3.1)
DGC	63.8±1.0	50.5±13.	26.0±0.9	88.6±1.0	45.3±1.3	96.2±0.4	54.0±0.6	60.6 (24.6)	8.3 (5.9)
S^2GC	79.9±0.6	38.5±12.	25.4±0.9	88.4±1.0	67.9±1.5	95.9±0.6	78.0±0.5	67.7 (26.2)	7.4 (3.4)
G^2CN	25.2±0.3	24.2±1.1	25.0±0.1	88.5±1.0	88.6±1.2	24.3±1.1	50.7±31.	46.6 (30.2)	11.6 (6.3)
GCN	36.3±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	64.8 (22.1)	8.1 (3.0)
SAGE	80.3±1.1	31.1±0.7	34.6±2.1	83.9±0.8	81.3±0.7	94.4±0.5	94.4±0.9	71.4 (27.0)	5.7 (2.9)
GCNII	73.5±1.2	30.7±0.7	27.1±1.3	84.2±0.8	69.0±1.4	90.6±0.9	80.4±1.2	65.1 (25.7)	8.7 (1.8)
H^2GCN	80.2±1.5	27.0±1.0	27.5±0.8	78.0±0.9	74.6±1.3	91.9±0.7	92.2±0.9	67.3 (28.2)	8.0 (3.9)
APPNP	66.0±2.6	30.3±1.2	25.2±0.7	71.2±4.9	43.8±2.0	83.2±3.8	58.7±4.5	54.1 (21.6)	12.9 (2.0)
GPR-GNN	73.4±0.4	74.6±0.7	65.9±2.1	89.9±0.6	87.6±1.2	95.0±1.1	91.9±1.1	82.6 (11.2)	3.3 (2.1)
GAT	32.7±5.5	42.6±4.8	36.8±5.7	64.0±5.7	55.6±6.8	68.5±7.1	67.0±12.	52.5 (15.0)	11.6 (4.1)
SLIMG (Ours)	81.0±1.1	87.1±1.4	89.2±1.2	88.1±0.5	88.9±0.7	94.4±0.6	93.9±0.5	88.9 (4.5)	2.6 (1.8)

The random case means that each feature element x_{ij} is determined independently of all other variables in the graph, providing no useful information. The semantic case represents a typical graph where **X** provides useful information of **y**. In this case, the feature $\mathbf{x}_i$ of each node i is directly correlated with the label y_i. In the structural case, **X** provides information of the graph structure, rather than the labels. Thus, **X** is meaningful for classification if **A** is not *uniform*, as it gives only indirect information of **y**.

5.2 Observations from Sanity Checks

Table 2 shows the results of sanity checks for our SLIMG and all baseline models whose details are given in Section 6.1. We assume 4 target classes of nodes, making the accuracy of random guessing 25%. Among the nine cases in Table 4a, we do not report the results on two cases where the graph does not give any information (i.e., "none helps"), since all methods produce similar accuracy.

SLIMG wins on all sanity checks, thanks to its careful design for the robustness to various graph scenarios. Although homophily **A** and useful (i.e., not random) **X** is a common assumption in many datasets, many nonlinear GNNs show failure (i.e., red cells) in such cases. This implies that the *theoretical expressiveness* of a model is often not aligned with its actual performance even in controlled testbeds, as we also show in our intuitive example in Figure 3. Only a few baselines succeed in other scenarios with different **A** and **X**, as we present in the observations below.

Observation 1 (Summary)**.** SLIMG wins on all sanity checks without failures (i.e., no red cells). SLIMG shows the best average accuracy and the average rank compared to 15 competitors.

Observation 2 (No network effects)**.** Only a few models including S^2GC and GraphSAGE perform well in uniform **A**, where the graph structure provides no useful information, since they have the raw **X** (not multiplied with **A**) in their propagator functions $\mathcal{P}$.

Observation 3 (Useless features)**.** None of the existing models in Table 2 succeeds with useless (i.e., random) features **X**, since their propagator functions $\mathcal{P}$ rely on **X** in all cases.

Observation 4 (Heterophily graphs)**.** Models that can utilize even-hop neighbors, such as G^2CN, GraphSAGE, and GPR-GNN, succeed in heterophily graphs either with semantic or structural **X**.

6 EXPERIMENTS

We conduct experiments on 13 real-world datasets to answer the following research questions (RQ):

- RQ1. **Accuracy:** How well does SLIMG work for semi-supervised node classification on real-world graphs?
- RQ2. **Success of simplicity:** How does SLIMG succeed even with its simplicity? What if we add nonlinearity to SLIMG?
- RQ3. **Speed and scalability:** How fast and scalable is SLIMG to large real-world graphs?
- RQ4. **Interpretability:** How to explain the importance of graph signals through the learned weights of SLIMG?
- RQ5. **Ablation study:** Are all the design decisions of SLIMG, such as two-hop aggregation, effective in real-world graphs?

6.1 Experimental Setup

We introduce our experimental setup including datasets, evaluation processes, and baselines for node classification.

Table 3: SLIMG wins most of the times on 13 real-world datasets (7 homophily and 6 heterophily graphs) against 15 competitors. We color the best and worst results as green and red, respectively, as in Table 2. SLIMG is the only approach that exhibits no failures (i.e., no red cells) in all datasets. Most competitors cause out-of-memory (OOM) errors on large graphs.

Model	Cora	CiteSeer	PubMed	Comp.	Photo	ArXiv	Products	Cham.	Squirrel	Actor	Penn94	Twitch	Pokec	Avg. Rank
LR	51.5±1.2	52.9±4.5	79.9±0.5	73.9±1.2	79.3±1.5	48.3±1.9	56.4±0.5	24.9±1.7	26.7±1.9	27.8±0.8	63.5±0.5	53.0±0.1	61.3±0.0	11.7 (4.2)
Reg. Kernel	67.8±2.5	62.1±4.4	83.4±1.4	80.3±1.4	87.1±1.2	O.O.M.	O.O.M.	29.4±2.6	24.3±2.3	29.6±1.4	O.O.M.	O.O.M.	O.O.M.	12.2 (3.8)
Diff. Kernel	70.6±1.5	62.7±3.8	82.1±0.4	83.1±1.0	89.8±0.6	O.O.M.	O.O.M.	34.5±7.9	28.3±1.5	24.7±0.9	53.5±0.8	O.O.M.	O.O.M.	11.8 (2.5)
RW Kernel	72.7±1.7	64.1±3.9	83.1±0.7	84.2±0.7	90.6±0.7	63.2±0.2	74.2±0.0	34.9±3.5	25.0±1.6	26.4±1.1	63.1±0.7	57.6±0.1	59.5±0.0	8.3 (3.3)
SGC	76.2±1.1	65.8±3.9	84.1±0.8	83.7±1.6	90.1±0.9	65.0±3.4	74.6±5.1	38.1±4.5	33.1±1.0	24.6±0.8	64.0±1.1	56.5±0.1	69.8±0.0	6.6 (4.2)
DGC	77.8±1.4	66.1±4.2	84.3±0.6	83.9±0.7	90.4±0.2	65.2±4.0	68.7±13.	37.2±3.7	29.2±1.2	25.2±2.1	62.5±0.4	58.2±0.2	60.7±0.1	6.6 (3.2)
S^2GC	78.3±1.5	66.9±4.4	84.3±0.3	83.1±0.8	90.1±0.8	62.0±7.4	58.3±18.	34.9±4.9	27.6±1.8	26.7±1.8	63.1±0.5	58.7±0.1	61.2±0.0	6.6 (2.7)
G^2CN	76.6±1.5	64.2±3.3	81.4±0.6	82.8±1.6	88.8±0.5	O.O.M.	O.O.M.	40.7±2.9	32.1±1.5	24.3±0.5	O.O.M.	O.O.M.	O.O.M.	10.5 (4.5)
GCN	76.0±1.2	65.0±2.9	84.3±0.5	85.1±0.9	91.6±0.5	62.8±0.6	O.O.M.	38.5±3.0	31.4±1.8	26.8±0.4	62.9±0.7	57.0±0.1	63.9±0.4	6.3 (2.4)
SAGE	74.6±1.3	63.7±3.6	82.9±0.4	83.8±0.5	90.6±0.5	61.5±0.6	O.O.M.	39.8±4.3	27.0±1.3	27.8±0.9	O.O.M.	56.6±0.4	68.9±0.1	8.5 (3.5)
GCNII	77.8±1.7	63.4±3.0	84.9±0.8	82.3±1.8	90.8±0.6	45.7±0.5	O.O.M.	30.5±2.5	21.9±3.0	29.0±1.3	64.5±0.5	56.9±0.6	62.1±0.3	8.4 (4.6)
H^2GCN	77.6±0.9	64.7±3.8	85.4±0.4	49.5±16.	75.8±11.	O.O.M.	O.O.M.	31.9±2.6	25.0±0.5	28.9±0.6	63.9±0.4	58.7±0.0	O.O.M.	8.9 (4.9)
APPNP	80.0±0.6	67.1±2.8	84.6±0.5	84.2±1.7	92.5±0.3	53.4±1.3	O.O.M.	30.9±4.7	23.9±3.2	26.1±1.0	63.7±0.9	47.3±0.3	57.4±0.4	7.6 (4.8)
GPR-GNN	78.8±1.3	64.2±4.0	85.1±0.7	85.0±1.0	92.6±0.3	58.5±0.8	O.O.M.	31.7±4.7	26.2±1.6	29.5±1.1	64.5±0.4	57.6±0.2	67.6±0.1	5.4 (3.7)
GAT	78.2±1.2	65.8±4.0	83.6±0.2	85.4±1.4	91.7±0.5	58.2±1.0	O.O.M.	39.1±4.1	28.6±0.6	26.4±0.4	60.5±0.8	O.O.M.	O.O.M.	7.5 (3.7)
SLIMG	77.8±1.1	67.1±2.3	84.6±0.5	86.3±0.7	91.8±0.5	66.3±0.3	84.9±0.0	40.8±3.2	31.1±0.7	30.9±0.6	68.2±0.6	59.7±0.1	73.9±0.1	1.9 (1.5)

Table 4: Dataset statistics. The first 7 datasets are homophily, and the last 6 are heterophily graphs.

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	5,429	1433	7
CiteSeer	3,327	4,732	3703	6
PubMed	19,717	44,338	500	3
Computers	13,752	245,861	767	10
Photo	7,650	119,081	745	8
ogbn-arXiv	169,343	1,166,243	128	40
ogbn-Products	2,449,029	61,859,140	100	30
Chameleon	2,277	36,101	2325	5
Squirrel	5,201	216,933	2089	5
Actor	7,600	29,926	931	5
Penn94	41,554	1,362,229	4814	2
Twitch	168,114	6,797,557	7	2
Pokec	1,632,803	30,622,564	65	2

Datasets. We use 7 homophily and 6 heterophily graph datasets in experiments, which were commonly used in previous works on node classification [4, 21, 23]. Table 4 shows a summary of dataset information. Cora, CiteSeer, and PubMed [26, 35] are homophily citation graphs between research articles. Computers and Photo [27] are homophily Amazon co-purchase graphs between items. ogbn-arXiv and ogbn-Products are large homophily graphs from Open Graph Benchmark [10]. Since we use only 2.5% of all labels as training data, we omit the classes with instances fewer than 100. Chameleon and Squirrel [24] are heterophily Wikipedia web graphs. Actor [29] is a heterophily graph connected by the co-occurrence of actors on Wikipedia pages. Penn94 [21, 30] is a heterophily graph of gender relations in a social network. Twitch [25] and Pokec [18] are large graphs, which have been relabeled by [21] to be heterophily. We make the heterophily graphs undirected as done in [4].

Evaluation. We perform semi-supervised node classification by randomly dividing all nodes in a graph by the 2.5%/2.5%/95% ratio into training, validation, and test sets. In this setting [4, 6], which is common in real-world data where labels are often scarce and expensive, we can properly evaluate the performance of each method in semi-supervised learning. We perform five runs of each experiment with different random seeds and report the average and standard deviation. All hyperparameter searches and early stopping are done based on validation accuracy for each run.

Competitors and hyperparameters. We include various types of competitors: linear models (LR, SGC, DGC, S^2GC, and G^2CN), coupled nonlinear models (GCN, GraphSAGE, GCNII, and H_2GCN), decoupled models (APPNP and GPR-GNN), and attention-based models (GAT). We perform row-normalization on the node features as done in most studies on GNNs. We search their hyperparameters for every data split through a grid search as described in Appendix D. The hidden dimension size is set to 64, and the dropout probability is set to 0.5. For the linear models, we use L-BFGS to train 100 epochs with the patience of 5. For the nonlinear ones, we use ADAM and update them for 1000 epochs with the patience of 200.

We also include three graph kernel methods [28], namely Regularized Laplacian, Diffusion Process, and the K-step Random Walk. They focus on feature transformation and are used with the LR classifier as SLIMG is. Thus, they perform as the direct competitors of SLIMG that apply different feature transformations. The propagator functions of graph kernel methods [28] are given as follows:

$$\text{(Reg. Kernel) } \mathcal{P}(\mathbf{A}, \mathbf{X}) = (\mathbf{I}_n + \sigma^2 \tilde{\mathbf{L}})^{-1}\mathbf{X} \tag{4}$$

$$\text{(Diff. Kernel) } \mathcal{P}(\mathbf{A}, \mathbf{X}) = \exp(-\sigma^2/2\tilde{\mathbf{L}})\mathbf{X} \tag{5}$$

$$\text{(RW Kernel) } \mathcal{P}(\mathbf{A}, \mathbf{X}) = (a\mathbf{I}_n - \tilde{\mathbf{L}})^p\mathbf{X}, \tag{6}$$

where $\tilde{\mathbf{L}} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{A})\mathbf{D}^{-1/2}$ is the normalized Laplacian matrix, and $\sigma = 1$, $a = 1$, and $p = 2$ are their hyperparameters.

SLIMG contains two hyperparameters, which are the weight of LASSO and the weight of group LASSO. It is worth noting that SLIMG does not need to recompute the features when searching the hyperparameters, while most of the linear methods need to do so because of including one or more hyperparameters in $\mathcal{P}$.

Table 5: SLIMG effectively combines different components. SLIMG-Ci represents that we use only the i-th component shown in Equation 3. Node classification is done accurately even we use a single component at each time, and SLIMG outperforms the best accuracy of a single component in 11 out of the 13 datasets. Green (■, ■) marks the top two.

Model	Cora	CiteSeer	PubMed	Comp.	Photo	ArXiv	Products	Cham.	Squirrel	Actor	Penn94	Twitch	Pokec
SLIMG-C1	46.3±3.0	29.2±2.5	64.5±1.0	77.6±1.0	78.5±0.9	51.4±0.2	73.6±2.9	41.9±2.0	29.1±1.0	21.6±1.2	60.9±0.6	59.3±0.1	66.7±0.0
SLIMG-C2	53.5±1.5	53.6±3.6	79.3±0.3	74.5±1.1	81.4±0.9	49.8±0.2	57.4±0.1	25.1±1.5	21.8±0.9	29.9±2.1	62.3±0.5	53.0±0.1	61.1±0.0
SLIMG-C3	77.6±0.7	62.7±4.3	77.4±0.8	86.0±1.0	90.3±0.8	66.2±0.2	82.5±0.0	40.6±1.0	27.6±3.2	24.1±1.4	64.7±0.6	53.1±0.1	73.2±0.1
SLIMG-C4	76.8±0.9	64.7±3.6	82.1±0.7	85.3±1.2	90.9±0.7	65.3±0.3	78.3±0.1	40.4±2.2	31.6±1.4	23.7±1.4	64.3±0.7	56.3±0.1	68.4±0.2
SLIMG	77.8±1.1	67.1±2.3	84.6±0.5	86.3±0.7	91.8±0.5	66.3±0.3	84.9±0.0	40.8±3.2	31.1±0.7	30.9±0.6	68.2±0.6	59.7±0.1	73.9±0.1

Table 6: Linearity is enough for SLIMG: it outperforms its own variants with nonlinearity that replace the linear classifier or the PCA function g with a nonlinear neural network. Green (■, ■) marks the top two.

Model	Cora	CiteSeer	PubMed	Comp.	Photo	ArXiv	Products	Cham.	Squirrel	Actor	Penn94	Twitch	Pokec
w/ MLP-2	65.9±1.1	54.2±5.3	83.3±0.2	84.8±0.5	90.1±1.9	65.8±0.1	85.7±0.0	40.0±2.5	30.5±0.5	28.8±1.0	66.7±1.7	60.3±0.3	76.5±0.1
w/ MLP-3	66.1±2.0	50.3±3.6	80.9±0.9	85.2±0.8	90.0±0.8	63.0±0.1	84.9±0.9	38.5±5.5	30.9±0.7	28.9±1.2	65.1±0.6	60.3±0.2	76.4±0.2
w/ NL Trans.	70.7±2.3	57.5±5.1	81.0±0.4	71.4±10.	77.9±2.2	57.0±0.6	O.O.M.	41.3±3.2	30.0±1.6	27.6±2.4	61.8±1.6	61.5±0.3	75.7±0.5
SLIMG	77.8±1.1	67.1±2.3	84.6±0.5	86.3±0.7	91.8±0.5	66.3±0.3	84.9±0.0	40.8±3.2	31.1±0.7	30.9±0.6	68.2±0.6	59.7±0.1	73.9±0.1

6.2 Accuracy (RQ1)

In Table 3, SLIMG is compared against 15 competitors on 13 real-world datasets (7 homophily and 6 heterophily graphs). We color the best and worst results as green and red, respectively, as in Table 2. We report the accuracy in Table 3 where ■, ■, ■ represent the top three methods (higher is darker), SLIMG outperforms all competitors in 4 homophily and 5 heterophily graphs, and achieves competitive accuracy in the rest; SLIMG is among the top three in 10 out of 13 times. Moreover, SLIMG is the only model that exhibits no failures (i.e., no red cells), and shows the best average rank with a significant difference from the second-best. It is notable that many competitors, even linear models such as the kernel methods and G^2CN, run out of memory in large graphs. This shows that linearity is not a sufficient condition for efficiency and scalability, and thus a careful design of the propagator function $\mathcal{P}$ is needed as in SLIMG.

6.3 Success of Simplicity (RQ2)

We conduct studies to better understand how SLIMG exhibits the superior performance on real-world graphs even with its simplicity. Tables 5 and 6 demonstrate the strength of simplicity for semi-supervised node classification in real-world graphs.

Flexibility. Table 5 illustrates the accuracy of SLIMG when only one of its four components in Equation 3 is used at each time. The accuracy of SLIMG with only a single component is higher than those of most baselines in Table 3 when an appropriate component is picked for each dataset, e.g., C1 for Twitch, C2 for Actor, C3 for Cora, and C4 for CiteSeer. This shows that high accuracy in semi-supervised node classification can be achieved by a well-designed simple model even without high *expressivity*. SLIMG focuses on the best component in each dataset effectively, improving the accuracy of individual components in 11 out of 13 datasets.

Nonlinearity. Many recent works on linear GNNs have shown that nonlinearity is not an essential component in semi-supervised node classification [20, 32, 39]. To support the success of SLIMG, we design three nonlinear variants of it:

- **w/ MLP-2:** We replace LR with a 2-layer MLP.
- **w/ MLP-3:** We replace LR with a 3-layer MLP.
- **w/ Nonlinear (NL) Transformation:** We replace the PCA function $g(\cdot)$ as a nonlinear function. Specifically, we adopt a 2-layer MLP for the first two components and a 2-layer GCN for the last two components. The transformed features are concatenated and given to another 2-layer MLP.

We use dropout with a probability of 0.5 to prevent overfitting in both MLP and GCN. The nonlinear models are trained with the same setting as GCN reported in Table 3. We report the result in Table 8, showing that adding nonlinearity does not necessarily improve the accuracy while sacrificing both scalability and interpretability.

Table 7: SLIMG is scalable. We measure the runtime of SLIMG by varying the numbers of features and edges in a graph.

Number of features	20	40	60	80	100
Time (s)	13.0	16.2	18.6	22.9	27.8
Number of edges	12M	25M	60M	80M	100M
Time (s)	12.3	19.9	21.9	24.0	27.8

6.4 Speed and Scalability (RQ3)

We plot the training time versus the accuracy of each model on the ogbn-arXiv, ogbn-Products, and Pokec graphs, which are the largest in our benchmark, in Figure 2. We report the training time of each model with the hyperparameters that show the highest validation accuracy. SLIMG achieves the highest accuracy in the ogbn-arXiv and ogbn-Products datasets while being 10.4× and 2.5× faster than the second-best model, respectively. SLIMG also shows the highest accuracy in the Pokec dataset, while being 18.0× faster than the best-performing deep model. It is worth noting that SLIMG is even faster than LR in ogbn-arXiv, taking fewer iterations than in LR during the optimization. Its fast convergence is owing to the orthogonalization of each component of the features.

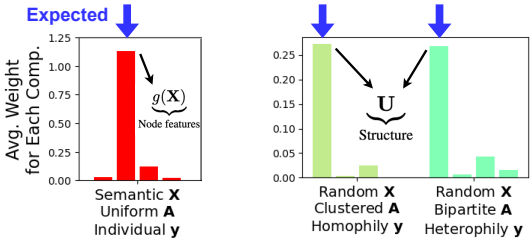
Table 7 shows the training time of SLIMG in the ogbn-Products graph with varying numbers of features and edges. Smaller graphs are created by removing edges uniformly at random. SLIMG scales well with both variables even in large graphs containing up to 61M edges, showing linear complexity as we claim in Lemma 2.

Table 8: Ablation study - SLIMG works best with the current design. Each design decision of SLIMG leads to an improvement of accuracy in real-world graphs; SLIMG is among the top two in all datasets, marked as green (,).

Model	Cora	CiteSeer	PubMed	Comp.	Photo	ArXiv	Products	Cham.	Squirrel	Actor	Penn94	Twitch	Pokec
w/o Sp. Reg.	77.8±0.6	65.0±3.5	83.8±0.5	85.9±0.8	91.7±0.7	65.2±0.2	83.4±1.9	40.1±3.8	30.7±1.0	30.1±0.6	67.4±0.6	59.8±0.1	74.2±0.0
w/o PCA	74.8±1.5	66.0±3.1	84.7±0.5	84.4±1.1	90.3±0.7	60.8±0.2	84.5±0.0	41.3±2.0	31.8±1.1	27.3±1.1	67.7±0.7	59.1±0.2	72.8±0.1
w/o Struct. U	78.1±1.0	67.4±2.5	84.4±0.3	86.0±0.5	92.1±0.4	66.1±0.2	82.5±0.6	37.5±4.2	29.8±0.4	31.3±0.5	65.8±0.6	56.8±0.0	72.8±0.1
SLIMG	77.8±1.1	67.1±2.3	84.6±0.5	86.3±0.7	91.8±0.5	66.3±0.3	84.9±0.0	40.8±3.2	31.1±0.7	30.9±0.6	68.2±0.6	59.7±0.1	73.9±0.1

Table 9: Ablation study - Two-step aggregation is good enough for SLIMG. The values k_{row} and k_{sym} represent the numbers of propagation steps for the $\mathbf{A}_{\text{row}}$ and $\tilde{\mathbf{A}}_{\text{sym}}$ components in Equation 3, respectively. We observe no consistent improvement of accuracy by increasing the values of k_{row} and k_{sym}, supporting the current design of SLIMG. Green (,) marks the top two.

k_{row}	k_{sym}	Cora	CiteSeer	PubMed	Comp.	Photo	ArXiv	Products	Cham.	Squirrel	Actor	Penn94	Twitch	Pokec
{2, 4, 6}	{2, 3, 4}	79.4±1.1	66.0±4.4	84.4±0.4	85.9±0.5	91.3±0.5	67.9±0.2	84.8±2.1	41.0±3.9	31.4±0.6	29.8±0.6	68.2±0.6	60.7±0.1	76.6±0.2
{2, 4}	{2, 3}	78.9±0.9	66.9±2.5	84.0±0.5	86.2±0.7	91.5±0.5	67.4±0.1	73.9±23.	41.4±4.0	31.0±0.7	30.4±0.4	68.3±0.5	60.8±0.1	76.0±0.1
6	4	79.2±0.7	66.2±3.4	84.0±0.3	85.2±0.7	91.0±0.8	67.3±0.2	84.7±1.7	38.2±6.3	29.2±1.5	31.2±0.8	67.7±0.7	59.8±0.1	74.2±0.2
4	3	79.2±0.8	66.1±3.5	84.2±0.5	85.8±0.6	91.3±0.4	67.2±0.1	85.4±0.0	38.6±7.1	29.5±1.8	30.4±0.7	67.5±0.6	60.0±0.1	74.6±0.1
2 (ours)	2 (ours)	77.8±1.1	67.1±2.3	84.6±0.5	86.3±0.7	91.8±0.5	66.3±0.3	84.9±0.0	40.8±3.2	31.1±0.7	30.9±0.6	68.2±0.6	59.7±0.1	73.9±0.1



(a) No network effects **(b) Useless features**

Figure 5: SLIMG is interpretable: it suppresses useless information and focuses on informative ones for each scenario: (a) self-features $g(\mathbf{X})$ and (b) structural features U. The learned weights are directly matched with the expectations.

6.5 Interpretability (RQ4)

Figure 5 illustrates the weights learned by the classifier in SLIMG for the sanity checks, where the ground truths are known. SLIMG assigns large weights to the correct factors in graphs with different mutual information between variables. When there are no network effects in Figure 5a, it successfully assigns the largest weights to the self-features $g(\mathbf{X})$, ignoring all other components. When the features are useless in Figure 5b, it puts most of the attention on the structural features **U**, which does not rely on **X**.

6.6 Ablation Studies (RQ5)

We run two types of ablation studies to support the main ideas of SLIMG: its design decisions and two-hop aggregation.

Design decisions. In Table 8, we show the accuracy of SLIMG when each of its core ideas is disabled: sparse regularization, PCA, and the structural features **U**. SLIMG performs best with all of its ideas enabled; it is always included in the top two in each dataset. This shows that SLIMG is designed effectively with ideas that help improve its accuracy on real-world datasets. Note that the sparse regularization and PCA improve the efficiency of SLIMG, by reducing the number of parameters, as well as its accuracy.

Receptive fields. To analyze the effect of changing the receptive field of SLIMG, we vary the distance of aggregation (i.e., the value of K) in Table 9. The values of k_{row} or k_{sym} given as sets, e.g., $\{2, 4, 6\}$, represent that we include more than one component to SLIMG with different K, increasing the overall complexity of decisions. Since $\mathbf{A}_{\text{row}}$ is designed to consider heterophily relations, we use only the even values of k_{row}. Table 9 shows no significant gain in accuracy by increasing the values of k_{row} and k_{sym}, or even including more components to SLIMG; the accuracies of all variants are similar to each other in all cases. That is, SLIMG works sufficiently well even with the 2-step aggregation for both $\mathbf{A}_{\text{row}}$ and $\tilde{\mathbf{A}}_{\text{sym}}$.

7 CONCLUSION

We propose SLIMG, a simple but effective model for semi-supervised node classification, which is designed by the *careful simplicity* principle. We summarize our contributions as follows:

- **C1 - Method:** SLIMG outperforms state-of-the-art GNNs in both synthetic and real datasets, showing the best robustness; SLIMG succeeds in all types of graphs with homophily, heterophily, noisy features, etc. SLIMG is scalable to million-scale graphs, even when other baselines run out of memory, making interpretable decisions from its linearity.
- **C2 - Explanation:** Our GNNEXP framework illuminates the fundamental similarities and differences of popular GNN variants (see Table 1), revealing their pain points.
- **C3 - Sanity checks:** Our sanity checks immediately highlight the strengths and weaknesses of each GNN method before it is sent to production (see Table 2).
- **C4 - Experiments:** Our extensive experiments explain the success of SLIMG in various aspects: linearity, robustness, receptive fields, and ablation studies (see Tables 3 to 9).

Reproducibility. Our code, along with our datasets for *sanity checks*, is available at https://github.com/mengchillee/SlimG.

ACKNOWLEDGMENTS

This work was partially funded by project AIDA (reference POCI-01-0247-FEDER-045907) under CMU Portugal, and a gift from PNC.

REFERENCES

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

[2] Shaked Brody, Uri Alon, and Eran Yahav. 2022. How Attentive are Graph Attention Networks?. In *ICLR*.

[3] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *ICML*.

[4] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *ICLR*.

[5] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*.

[6] Kaize Ding, Jianling Wang, James Caverlee, and Huan Liu. 2022. Meta Propagation Networks for Graph Few-shot Semi-supervised Learning. In *AAAI*. AAAI Press, 6524–6531. https://ojs.aaai.org/index.php/AAAI/article/view/20605

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*.

[8] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.

[9] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NeurIPS*.

[10] 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).

[11] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. 2021. Combining Label Propagation and Simple Models out-performs Graph Neural Networks. In *ICLR*.

[12] Vassilis N. Ioannidis, Meng Ma, Athanasios N. Nikolakopoulos, and Georgios B. Giannakis. 2017. Kernel-based Inference of Functions over Graphs. *CoRR* abs/1711.10353 (2017).

[13] Dongkwan Kim and Alice Oh. 2021. How to Find Your Friendly Neighborhood: Graph Attention Design with Self-Supervision. In *ICLR*.

[14] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[15] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *ICLR*.

[16] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion Improves Graph Learning. In *NeurIPS*.

[17] Jure Leskovec, Deepayan Chakrabarti, Jon M. Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. 2010. Kronecker Graphs: An Approach to Modeling Networks. *J. Mach. Learn. Res.* 11 (2010), 985–1042.

[18] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection.

[19] Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs Go As Deep As CNNs?. In *ICCV*.

[20] Mingjie Li, Xiaojun Guo, Yifei Wang, Yisen Wang, and Zhouchen Lin. 2022. G^2CN: Graph Gaussian Convolution Networks with Concentrated Graph Filters. In *ICML*.

[21] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. 2021. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In *NeurIPS*.

[22] Meng Liu, Hongyang Gao, and Shuiwang Ji. 2020. Towards Deeper Graph Neural Networks. In *KDD*.

[23] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *ICLR*.

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

[25] Benedek Rozemberczki and Rik Sarkar. 2021. Twitch Gamers: A Dataset for Evaluating Proximity Preserving and Structural Role-Based Node Mmbeddings. *arXiv preprint arXiv:2101.03091* (2021).

[26] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Mag.* 29, 3 (2008), 93–106.

[27] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *CoRR* abs/1811.05868 (2018).

[28] Alexander J. Smola and Risi Kondor. 2003. Kernels and Regularization on Graphs. In *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2777)*, Bernhard Schölkopf and Manfred K. Warmuth (Eds.). Springer, 144–158.

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

[30] Amanda L Traud, Peter J Mucha, and Mason A Porter. 2012. Social Structure of Facebook Networks. *Physica A: Statistical Mechanics and its Applications* 391, 16 (2012), 4165–4180.

[31] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

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

[33] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*.

[34] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.* 32, 1 (2021), 4–24.

[35] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *ICML*.

[36] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *KDD*.

[37] Jaemin Yoo, Hyunsik Jeon, and U Kang. 2019. Belief Propagation Network for Hard Inductive Semi-Supervised Learning. In *IJCAI*.

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

[39] Hao Zhu and Piotr Koniusz. 2021. Simple Spectral Graph Convolution. In *ICLR*.

[40] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. In *NeurIPS*.

A PROOFS OF LEMMAS

A.1 Proof of Lemma 1

PROOF. We prove the lemma for each of SGC, DGC, S^2GC, and G^2CN. The propagator function of SGC [33] directly fits the definition of linearization. DGC [32] has variants DGC-Euler and DGC-DK. We focus on DGC-Euler, which is mainly used in their experiments. Then, DGC also fits the definition of linearization. S^2GC [39] computes the summation of features propagated with different numbers of steps. The original formulation divides the added features by K, which is safely ignored as we multiply the weight matrix $\mathbf{W}$ to the transformed feature for classification.

G^2CN [20] does not provide an explicit formulation of the propagator function. The parameterized version $\mathcal{P}'$ of the propagator function is $\mathcal{P}'(\mathbf{A}, \mathbf{X}; \{\theta_i\}_{i=1}^N) = \sum_{i=1}^N \theta_i \mathbf{H}_{i,K}$, where θ_i is a parameter. The k-th feature representation $\mathbf{H}_{i,k}$ is recursively defined as $\mathbf{H}_{i,k} = [\mathbf{I} - \frac{T_i}{K}((b_i - 1)\mathbf{I} + \mathbf{A}_{\text{sym}})^2]\mathbf{H}_{i,k-1}$, where $\mathbf{L} = \mathbf{I} - \mathbf{A}_{\text{sym}}$ is the normalized Laplacian matrix, N, T_i, and b_i are hyperparameters, and $\mathbf{H}_{i,0} = \mathbf{X}$. Then, we make it contain no learnable parameters as $\mathcal{P}(\mathbf{A}, \mathbf{X}) = \|_{i=1}^N \mathbf{H}_{i,K}$. We prove the lemma from the four cases. ∎

A.2 Proof of Lemma 2

PROOF. The training of SLIMG consists of three parts: SVD, PCA, and LR. The time complexity of SVD is $O(de + d^2 n)$ as SLIMG runs the sparse truncated SVD for generating structural features. The complexity of PCA, which is applied to each of the four components in Equation 3, is $O(d^2 n + d^3)$. The complexity of the gradient-based optimization of LR is $O(dnt)$, where t is the number of epochs. We safely assume $t < n$, since $t < 100$ in all our experiments. We prove the lemma by combining the three complexity terms. ∎

B LINEARIZATION PROCESSES

B.1 Linearization of Decoupled GNNs

Lemma 3. *PPNP [15] is linearized by GNNEXP as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = (\mathbf{I}_n - (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}})^{-1}\mathbf{X}, \tag{7}$$

where $0 < \alpha < 1$ controls the weight of self-loops.

PROOF. The proof is straightforward, since the authors present a closed-form representation of the propagator function. We remove the activation function from the original representation. ∎

Lemma 4. *APPNP [15] is linearized by GNNEXP as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \left[\sum_{k=0}^{K-1} \alpha(1-\alpha)^k \tilde{\mathbf{A}}_{\text{sym}}^k + (1-\alpha)^K \tilde{\mathbf{A}}_{\text{sym}}^K\right]\mathbf{X}, \tag{8}$$

where $0 < \alpha < 1$ controls the weight of self-loops.

PROOF. We assume that the initial node representation is created by a single linear layer, i.e., $\mathbf{XW}$. Then, the k-th representation matrix $\mathbf{H}_k$ is represented as $\mathbf{H}_k = (1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H}_{k-1} + \alpha\mathbf{XW}$, where $0 < \alpha < 1$ is a hyperparameter. We derive the closed-form representation of $\mathbf{H}_K$ and remove redundant parameters. ∎

Lemma 5. *GDC [16] is linearized by GNNEXP as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \tilde{\mathbf{S}}'_{\text{sym}} \odot \mathbf{1}(\tilde{\mathbf{S}}'_{\text{sym}} \geq \epsilon), \tag{9}$$

where $\mathbf{S} = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k \tilde{\mathbf{A}}_{\text{sym}}^k$, $\odot$ is the elementwise multiplication, $\mathbf{1}$ is a matrix that contains one if each element holds the condition and zero if otherwise, and α and ϵ are hyperparameters.

PROOF. GDC presents various forms of propagation functions by generalizing APPNP. We pick the most representative one given in the paper, which is directly related to APPNP. The unnormalized version of the propagation matrix is $\mathbf{S}' = \sum_{k=0}^{\infty} \alpha(1-\alpha)^k \tilde{\mathbf{A}}_{\text{sym}}^k$, which is then normalized and sparsified as $\mathbf{S} = \text{sparsify}(\tilde{\mathbf{S}}'_{\text{sym}})$. The matrix $\tilde{\mathbf{S}}'_{\text{sym}}$ 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 ϵ, which is a hyperparameter, and b) selecting the top k neighbors for each node. We take the first approach, which is easier to represent in a closed form, getting Equation 9. ∎

Lemma 6. *GPR-GNN [4] is linearized by GNNEXP as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \|_{k=0}^{K} \tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X}. \tag{10}$$

PROOF. We assume that the initial node representation is created by a single linear layer, i.e., $\mathbf{XW}$. Then, we replace the summation in the original propagator function with concatenation to remove the learnable parameters, getting Equation 10. ∎

B.2 Linearization of Coupled GNNs

Lemma 7. *ChebNet [5] is linearized by GNNEXP as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \|_{k=0}^{K-1} \mathbf{A}_{\text{sym}}^k \mathbf{X}. \tag{11}$$

PROOF. 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 θ is $\mathcal{P}'(\mathbf{A}, \mathbf{X}; \theta) = \sum_{k=0}^{K-1} \theta_k \mathbf{H}_k$ where $\mathbf{H}_k = a_k \mathbf{A}_{\text{sym}}^k \mathbf{X} + a_{k-1}\mathbf{A}_{\text{sym}}^{k-1}\mathbf{X} + \cdots + a_0\mathbf{X}$, and $a_0, \cdots, a_k$ are constants. Since we have K free parameters $\theta_0, \cdots, \theta_K$, we safely rewrite the propagator function as $\mathcal{P}'(\mathbf{A}, \mathbf{X}; \theta) = \sum_{k=0}^{K-1} \theta_k \mathbf{A}_{\text{sym}}^k \mathbf{X}$. We remove the parameters by replacing the summation with concatenation. ∎

Lemma 8. *GraphSAGE [9] is linearized by GNNEXP as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \|_{k=0}^{K} \mathbf{A}_{\text{row}}^k \mathbf{X}. \tag{12}$$

PROOF. We assume the mean aggregator of GraphSAGE among various choices. By removing the activation function, each layer of GraphSAGE is linearized as $\mathcal{F}(\mathbf{X}) = \mathbf{XW}_1 + \mathbf{A}_{\text{row}}\mathbf{XW}_2$, 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. If we stack K layers with reparametrization, we get $\mathcal{F}^K(\mathbf{X}) = \sum_{k=1}^{K} \mathbf{A}_{\text{row}}^k \mathbf{XW}_k$. 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. ∎

Lemma 9. *GCNII [3] is linearized by GNNEXP as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \|_{k=0}^{K-2} \tilde{\mathbf{A}}_{\text{sym}}^k \mathbf{X} \,\|\, ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}^K + \alpha\tilde{\mathbf{A}}_{\text{sym}}^{K-1})\mathbf{X}, \tag{13}$$

where α is a hyperparameter.

PROOF. After removing the activation function, the l-th layer $\mathcal{F}_l$ of GCNII is $\mathcal{F}_l(\mathbf{H}) = ((1-\alpha_l)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha_l\mathbf{X}))((1-\beta_l)\mathbf{I} + \beta_l\mathbf{W}_l)$, where α_l and β_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 β_l, since $\mathbf{W}_l$ is a free parameter. We also set α_l to a constant α which is the same for every layer l, following the original paper [3].[1] Then, the equation is simplified as $\mathcal{F}_l(\mathbf{H}) = ((1-\alpha)\tilde{\mathbf{A}}_{\text{sym}}\mathbf{H} + \alpha\mathbf{X})\mathbf{W}_l$. The

[1] α is set to 0.1 in the original paper of GCNII [3].

Table 10: Search space 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^2GC	$wd = [0, 5e^{-4}], K = 16, \alpha = [0.01, 0.03, 0.05, 0.07, 0.09]$
G^2CN	$wd = [0, 5e^{-4}], K = 100, N = 2, T_1 = T_2 = [10, 20, 30, 40], b_1 = 0, b_2 = 2$
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	$wd = [0, 5e^{-4}], lr = 0.01, K = [8, 16, 32, 64], \alpha = [0.1, 0.2, 0.5], \theta = [0.5, 1, 1.5]$
H_2GCN	$wd = [0, 5e^{-4}], lr = [2e^{-3}, 0.01, 0.05], K = [1, 2]$
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	$wd = [0, 5e^{-4}], lr = [2e^{-3}, 0.01, 0.05], K = 2, heads = 8$
SLIMG	$wd_1 = [1e^{-3}, 1e^{-4}, 1e^{-5}], wd_2 = [1e^{-3}, 1e^{-4}, 1e^{-5}, 1e^{-6}]$

closed-form representation is $\mathcal{F}_K(\mathbf{X}) = (1-\alpha)^{K-1}((1-\alpha)\tilde{\mathbf{A}}^K_{\text{sym}} + \alpha\tilde{\mathbf{A}}^{K-1}_{\text{sym}})\mathbf{X}\mathbf{W}_K + \alpha\sum_{k=0}^{K-2}(1-\alpha)^k\tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X}\mathbf{W}_k$. We safely remove the constants that can be included in the weight matrices, and replace the summation with concatenation. ■

Lemma 10. *H_2GCN [40] is linearized by GnnExp as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \|_{k=0}^{2K}\mathbf{A}^k_{\text{sym}}\mathbf{X}. \tag{14}$$

Proof. H_2GCN concatenates the features generated from every layer, each of which also concatenates the features averaged for the one- and two-hop neighbors from the previous layer. The self-loops are not added to $\mathbf{A}$ during the propagations. ■

B.3 Partial Linearization of Attention GNNs

Lemma 11. *DA-GNN [22] is partially linearized by GnnExp as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \sum_{k=0}^{K}\text{diag}(\tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X}\mathbf{w})\tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X}. \tag{15}$$

where $\text{diag}(\cdot)$ *is a function that generates a diagonal matrix from a vector, and* $\mathbf{w}$ *is a learnable parameter.*

Proof. DA-GNN has separate feature transformation and propagation steps as in the decoupled models. We assume that the initial node representation is created by a single linear layer, i.e., $\mathbf{X}\mathbf{W}$. Then, the k-th representation matrix is $\mathbf{H}_k = \tilde{\mathbf{A}}^k_{\text{sym}}\mathbf{X}\mathbf{W}$. 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: $\mathcal{P}(\mathbf{A}, \mathbf{X}) = \sum_{k=0}^{K}\text{diag}(\mathbf{H}_k\mathbf{s})\mathbf{H}_k$, where $\mathbf{s}$ is a learnable weight vector. Lastly, we remove the redundant parameters. ■

Lemma 12. *GAT [31] is partially linearized by GnnExp as*

$$\mathcal{P}(\mathbf{A}, \mathbf{X}) = \prod_{k=1}^{K}\left[\text{diag}(\mathbf{X}\mathbf{w}_{k,1})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\text{diag}(\mathbf{X}\mathbf{w}_{k,2})\right]\mathbf{X}, \tag{16}$$

where $\mathbf{w}_{k,1}$ *and* $\mathbf{w}_{k,2}$ *are learnable weight vectors.*

Proof. We apply the following changes to linearize GAT, whose linearization is not straightforward due to the nonlinearity in the attention function: (a) We simplify the attention function, removing the exponential and normalization: $\alpha_{ij} = \exp(e_{ij})/\sum_k \exp(e_{ik}) \approx e_{ij}$. (b) We remove LeaklyReLU in the computation of e_{ij}. (c) We assume the single-head attention. Then, the edge weight e_{ij}, which is the (i, j)-th element of the propagator matrix, is defined as $e_{ij} = \mathbf{a}^\top_{\text{dst}}(\mathbf{W}^\top\mathbf{x}_i) + \mathbf{a}^\top_{\text{src}}(\mathbf{W}^\top\mathbf{x}_j)$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are feature vectors of length d for node i and j, respectively, $\mathbf{W}$ is a $d \times c$ weight matrix, and $\mathbf{a}_{\text{dst}}$ and $\mathbf{a}_{\text{src}}$ are learnable weight vectors of length c.

We derive the initial form of a linearized GAT layer as $\mathbf{H} = [\text{diag}(\mathbf{X}\mathbf{W}\mathbf{a}_{\text{dst}})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\text{diag}(\mathbf{X}\mathbf{W}\mathbf{a}_{\text{src}})]\mathbf{X}\mathbf{W}$. Since all $\mathbf{a}_{\text{dst}}$, $\mathbf{a}_{\text{src}}$, and $\mathbf{W}$ are free parameters, we generalize it as $\mathbf{H} = [\text{diag}(\mathbf{X}\mathbf{w}_{\text{dst}})\tilde{\mathbf{A}} + \tilde{\mathbf{A}}\text{diag}(\mathbf{X}\mathbf{w}_{\text{src}})]\mathbf{X}\mathbf{W}$, where $\mathbf{w}_{\text{dst}}$ and $\mathbf{w}_{\text{src}}$ are learnable vectors that replace $\mathbf{a}_{\text{dst}}$ and $\mathbf{a}_{\text{src}}$, respectively. Then, we get Equation 16. ■

C IMPLEMENTATION OF SANITY CHECKS

There exist various ways to generate synthetic graphs satisfying the requirements of our sanity checks [1, 17]. However, we choose the simplest approach to focus on the mutual information between variables, rather than other characteristics of real-world graphs.

Structure. We assume that the number of node clusters is the same as the number c of labels. We divide all nodes into c groups and then decide the edge densities for intra- and inter-connections of groups based on the type: uniform, homophily, and heterophily. The expected number of edges is the same for all three cases.

A notable characteristic of a heterophily structure $\mathbf{A}$ is that $\mathbf{A}^2$ follows homophily. If we create inter-group connections for all pairs of different groups, it creates noisy $\mathbf{A}^2$ with inter-group connections. For 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 $\mathbf{A}$, as shown in Figure 4c.

Features. We assume that every feature element x_{ij} is basically sampled from a uniform distribution. In the random case, we sample each element from the distribution $\mathcal{U}(0, 1)$ between 0 and 1. In the structural case, we run the low-rank support vector decomposition (SVD) [8] to make $\mathbf{X}$ have structural information. Given $\mathbf{U}\Sigma\mathbf{V}^\top \approx \mathbf{A}$ from the low-rank SVD, we take $\mathbf{U}$ and normalize each column to have the zero-mean and unit-variance. The rank r in the SVD is a hyperparameter; higher r captures the structure better but can give noisy information. We also apply ReLU to make $\mathbf{U}$ positive.

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 $\mathbf{x}_i$ such that $\arg\max_k \mathbf{x}_i^\top\mathbf{v}_k = y$. In this way, we have random vectors that have sufficient semantic information for the classification of labels, with a guarantee that the perfect linear decision boundaries can be drawn in the feature space $\mathbf{X}$.

D HYPERPARAMETERS

Table 10 summarizes the search space of hyperparameters for SLIMG and the competitors. We conduct a grid search based on the validation accuracy for each data split for a fair comparison between different models. It is notable that we run all our experiments on five different random seeds, and thus the optimal set of hyperparameters can be found differently for those five runs.