

## A EXPERIMENTS OF PROP ON GRAPH CLASSIFICATION

**Methods.** We first aggregate node features within  $K$ -hop neighbors without any trainable weights, then pool aggregated node features into a global graph representation, *i.e.*,

$$\mathbf{H}_{\text{PROP}} = \frac{1}{N} \sum_i \mathbf{H}_i, \quad \mathbf{H} = \tilde{\mathbf{A}}^K \mathbf{X}, \quad (6)$$

where  $N$  is the number of nodes,  $\tilde{\mathbf{A}} = \mathbf{D}'^{-\frac{1}{2}} \mathbf{A}' \mathbf{D}'^{-\frac{1}{2}}$  with  $\mathbf{A}' = \mathbf{A} + \mathbf{I}$ .

**Datasets.** For the graph classification task, we choose molecules datasets MUTAG (Debnath et al., 1991) and NCI1 (Wale et al., 2008), bioinformatics datasets PROTEINS (Borgwardt et al., 2005), and DD (Dobson & Doig, 2003), social networks IMDB-BINARY, IMDB-MULTI (Yanardag & Vishwanathan, 2015), and COLLAB (Yanardag & Vishwanathan, 2015).

**Baselines.** We consider three categories of representative methods as baselines: 1) graph kernel methods including GL (Shervashidze et al., 2009), WL (Shervashidze et al., 2011), and DGK (Yanardag & Vishwanathan, 2015), 2) traditional graph embedding methods including node2vec (Grover & Leskovec, 2016), sub2vec (Adhikari et al., 2018), and graph2vec (Narayanan et al., 2017), 3) contrastive learning methods including InfoGraph (Sun et al., 2020), GraphCL (You et al., 2020), MVGRL (Hassani & Khasahmadi, 2020), JOAOv2 (You et al., 2021), ADGCL (Suresh et al., 2021).

**Settings.** Following (You et al., 2020), we train the model in an unsupervised manner and feed the learned representation into a downstream SVM classifier. To keep comparison fairness, we tune hyperparameters in a unified combination, and keep the search space among methods as consistent as possible. Details can be found in Appendix O.

**Results.** As shown in Table 9, although free of training, PROP surpasses most graph kernels and traditional embeddings, and performs comparably with GCL methods. On average, the mean performance gap between PROP and the best method across datasets is only 2.82%. The results show the **potential** of PROP on the graph classification task. **Notably, common graph classification benchmarks often have less informative node features than node classification benchmarks, even lacking node attribute description as seen in Table 24.** This probably impedes the ability of PROP. An optional choice is utilizing Laplacian positional embeddings or random-walk embeddings as widely discussed in the literature of graph Transforms (Yun et al., 2019; Ying et al., 2021; Rampásek et al., 2022). **We leave deeper research on graph classification tasks for future work.**

Table 9: Test accuracy (%) of graph classification benchmarks, comparing PROP and GSSL methods. The compared results are from published papers, and – indicates that results are unavailable. We report the performance gap between one method and the best method, averaged across datasets in the **Mean Gap**. column. **Red** indicates the best method, while underlined represents the second-best.

	PROTEINS	MUTAG	DD	NCI1	IMDB-B	IMDB-M	COLLAB	Mean Gap. ↓
<i>Graph Kernel</i>								
GL	–	81.66 ± 2.11	–	–	65.87 ± 0.98	–	–	7.60
WL	72.92 ± 0.56	80.72 ± 3.00	–	<u>80.01 ± 0.50</u>	72.30 ± 3.44	–	–	2.88
DGK	73.30 ± 0.82	87.44 ± 2.72	–	<b>80.31 ± 0.46</b>	66.96 ± 0.56	–	–	2.37
<i>Traditional Graph Embedding</i>								
node2vec	57.49 ± 3.57	72.63 ± 10.20	–	54.89 ± 1.61	–	–	–	16.61
sub2vec	53.03 ± 5.55	61.05 ± 15.80	–	52.84 ± 1.47	55.26 ± 1.54	–	–	19.79
graph2vec	73.30 ± 2.05	83.15 ± 9.25	–	73.22 ± 1.81	71.10 ± 0.54	–	–	3.54
<i>Graph Contrastive Learning</i>								
MVGRL	–	75.40 ± 7.80	–	–	63.60 ± 4.20	–	–	11.87
InfoGraph	<b>74.44 ± 0.31</b>	<u>89.01 ± 1.13</u>	72.85 ± 1.78	76.20 ± 1.06	<b>73.03 ± 0.87</b>	48.66 ± 0.67	70.65 ± 1.13	2.07
GraphCL	<u>74.39 ± 0.45</u>	86.80 ± 1.34	<b>78.62 ± 0.40</b>	77.87 ± 0.41	71.14 ± 0.44	<u>48.49 ± 0.63</u>	<u>71.36 ± 1.15</u>	<b>1.52</b>
JOAOv2	74.07 ± 1.10	87.67 ± 0.79	77.40 ± 1.15	78.36 ± 0.53	70.83 ± 0.25	–	69.33 ± 0.34	<u>1.78</u>
ADGCL	73.81 ± 0.46	<b>89.70 ± 1.03</b>	75.10 ± 0.39	69.67 ± 0.51	<u>72.33 ± 0.56</u>	<b>49.89 ± 0.66</b>	<b>73.32 ± 0.61</b>	2.21
<b>PROP</b>	71.07 ± 0.30	87.44 ± 1.53	<u>78.39 ± 0.37</u>	75.24 ± 0.14	71.22 ± 0.28	47.11 ± 0.18	69.07 ± 0.05	2.82

## B GRAPH STRUCTURE AS SUPERVISED SIGNAL

The taxonomy of homophily and heterophily is widely used to tell whether the graph structure is informative for training GCN-like models. Beyond the discussion on homophily and heterophily, recent metrics characterizing graphs are proposed and show closer relationships with the GNN performance (Mao et al., 2023; Luan et al., 2023; Platonov et al., 2023a). For example, Ma et al. (2021) claim that the inter-class similarity on Squirrel is slightly higher than the intra-class similarity for most classes, which substantiates the middling performance of GCN.

However, the performance of GCN-like models is an interplay between graph structure and node features. Therefore, a bad GCN performance can not indicate the helplessness of graph structure, or vice versa. For verification, we design experiments based on the mutual information of labels and different graph elements. To escape from the entanglement of structure and node features, we use MLP instead of GCN as the trainable model with node features  $\mathbf{X}$ , adjacency matrix  $\mathbf{A}$ , and the concatenation of the two as inputs, respectively. The correspondence is as follows:

- $I(\mathbf{Y}; \mathbf{X})$ : MLP with  $\mathbf{X}$  as inputs.
- $I(\mathbf{Y}; \mathbf{A})$ : MLP with  $\mathbf{A}$  as inputs.
- $I(\mathbf{Y}; \mathbf{X}; \mathbf{A})$ : MLP with  $[\mathbf{X}, \mathbf{A}]$  as inputs, where  $[\ ]$  denotes concatenation.

The results are shown in Table 10. It is surprising that **for some heterophily datasets, MLP with the graph structure as inputs gets satisfying performance**. For example, for the Squirrel dataset with a low homophily ratio of 0.22, MLP based on the graph structure achieves 73.58% accuracy. Therefore, even presenting a low homophily ratio, the graph structure can still serve as a highly qualified supervision signal for predicting labels.

Table 10: Test accuracy (%) of MLP with different input signals on node classification benchmarks.  $\mathcal{H}(G)$  denotes the edge homophily ratio introduced in Zhu et al. (2020a). Lower  $\mathcal{H}(G)$  denotes graphs with a high heterophily level. **Bold** indicates the best, while underlined represents the second-best choice.

	Cora	CiteSeer	PubMed	Chameleon	Squirrel	Actor
$\mathcal{H}(G)$	0.81	0.74	0.80	0.23	0.22	0.22
MLP( $\mathbf{X}$ )	73.64	<u>70.72</u>	<u>85.75</u>	49.34	35.06	<b>36.51</b>
MLP( $\mathbf{A}$ )	<u>78.27</u>	57.81	81.41	<b>77.41</b>	<b>73.58</b>	21.84
MLP( $[\mathbf{X}, \mathbf{A}]$ )	<b>82.29</b>	<b>73.57</b>	<b>85.83</b>	<u>71.05</u>	<u>67.63</u>	<u>31.84</u>

## C TRIALS IN FEW-SHOT LEARNING

In Section 5, we observe that GCL has the potential to learn good propagation coefficients. It inspires methods in the *few-shot* scenario, where a model is tasked with achieving effective generalization from a minimal number of labeled examples per class.

In this study, we examine the  $N$ -shot case where  $N$  support examples are used for training. As baselines, we evaluate the ChebNetII model trained with both supervised learning (SL) and contrastive learning (CL). As shown in Table 11, SL exhibits low accuracy due to sparse labeling, while CL performs relatively better, given access to all provided samples.

Based on our findings, we first train the ChebNetII model using contrastive learning. We then fix the propagation coefficients learned in GCL and focus on optimizing the transformation weights through a supervised objective. We term the method as **Fix-prop SL**. As illustrated in Table 11, this approach yields improvements on several benchmarks. For instance, Fix-prop SL enhances SL accuracy from 57.51% to 72.60% on Cora in the 5-shot case, and from 39.19% to 65.39% in the 3-shot case. The results demonstrate the potential of integrating SL and CL from a decoupling perspective in few-shot learning. However, the Fix-prop SL approach has minimal impact on the Squirrel and Chameleon datasets. It is important to note that we keep hyperparameters consistent across all training methods and benchmarks, leaving ample room for further exploration beyond this initial investigation.

Table 11: Test accuracy (%) of node classification benchmarks in the few-shot scenario. **Bold** indicates the best, while underlined represents the second-best choice.

	Training	Cora	CiteSeer	PubMed	Squirrel	Chameleon
5 Shot	SL	57.51 $\pm$ 2.29	43.11 $\pm$ 3.75	59.62 $\pm$ 2.56	20.15 $\pm$ 0.30	22.09 $\pm$ 1.60
	CL	<u>66.88 <math>\pm</math> 2.29</u>	<b>55.02 <math>\pm</math> 4.64</b>	<u>63.20 <math>\pm</math> 2.64</u>	<b>28.41 <math>\pm</math> 0.87</b>	<b>36.92 <math>\pm</math> 2.52</b>
	Fix-prop SL	<b>72.60 <math>\pm</math> 1.43</b>	<u>53.26 <math>\pm</math> 4.03</u>	<b>67.66 <math>\pm</math> 2.58</b>	<u>20.60 <math>\pm</math> 0.90</u>	<u>23.30 <math>\pm</math> 1.91</u>
3 Shot	SL	39.19 $\pm$ 3.96	37.52 $\pm$ 2.25	55.89 $\pm$ 2.55	20.27 $\pm$ 0.55	21.40 $\pm$ 1.26
	CL	64.46 $\pm$ 4.34	<b>55.85 <math>\pm</math> 5.15</b>	<u>59.88 <math>\pm</math> 3.49</u>	<b>25.89 <math>\pm</math> 1.54</b>	<b>36.12 <math>\pm</math> 1.34</b>
	Fix-prop SL	<b>65.39 <math>\pm</math> 2.15</b>	<u>46.90 <math>\pm</math> 3.40</u>	<b>61.46 <math>\pm</math> 5.49</b>	<u>20.38 <math>\pm</math> 0.69</u>	<u>27.85 <math>\pm</math> 3.02</u>

## D EXTENSIVE EXPERIMENTS OF SECTION 5.1

In Section 5.1, we show that in the GRACE method, after replacing the trained transformation weights with a random Gaussian matrix, the downstream performance does not deteriorate as expected. We conclude that the transformation weights learned in GCL are not better than random.

To enhance the generalizability of our conclusion, we extended our experimental evaluations to include additional GCL methods. The experimental settings are kept the same. Table 12 and Table 13 respectively show the results using the DGI and BGRL methods. For DGI, after replacing the transformation weights  $W_1$  or  $W_2$  with a random Gaussian matrix, the performance is comparable with before. Moreover, replacing both  $W_1$  and  $W_2$  raises the performance from 71.92% to 72.18% on average. For BGRL, substituting the original transformation weights with random matrices brings an increase of nearly 2% in average performance at best. Although we can not exhaustively try all GCL methods, the results of the representative methods are able to verify that GCL fails to learn effective transformation weights.

Table 12: Test accuracy (%) of node classification benchmarks, comparing the transformation weights ( $W_1$  and/or  $W_2$ ) learned in DGI with random weights. **Red** indicates the best method, while underlined represents the second-best choice.

Method	Cora	CiteSeer	PubMed	Squirrel	Chameleon	Texas	Wisconsin	Cornell	Mean
DGI	83.10 $\pm$ 1.10	<u>66.18 <math>\pm</math> 1.30</u>	82.47 $\pm$ 0.38	<b>41.55 <math>\pm</math> 0.78</b>	61.75 $\pm$ 1.64	<u>85.57 <math>\pm</math> 2.95</u>	<u>74.00 <math>\pm</math> 2.75</u>	80.82 $\pm$ 1.97	71.93
Randomize $W_1$	79.75 $\pm$ 0.80	65.59 $\pm$ 0.60	82.66 $\pm$ 0.39	38.65 $\pm$ 0.87	<u>66.04 <math>\pm</math> 0.85</u>	85.41 $\pm$ 1.97	<b>75.88 <math>\pm</math> 3.75</b>	<u>80.82 <math>\pm</math> 1.80</u>	71.85
Randomize $W_2$	<b>83.61 <math>\pm</math> 0.92</b>	<b>70.19 <math>\pm</math> 0.97</b>	82.56 $\pm$ 0.30	39.38 $\pm$ 1.09	60.20 $\pm$ 1.31	<b>85.74 <math>\pm</math> 3.11</b>	73.38 $\pm$ 1.63	<b>80.98 <math>\pm</math> 1.97</b>	<u>72.01</u>
Randomize both	80.99 $\pm$ 0.77	65.85 $\pm$ 0.60	<b>82.89 <math>\pm</math> 0.37</b>	<u>41.04 <math>\pm</math> 0.94</u>	<b>68.21 <math>\pm</math> 1.20</b>	84.92 $\pm$ 3.11	72.75 $\pm$ 1.00	80.82 $\pm$ 1.97	<b>72.18</b>

Table 13: Test accuracy (%) of node classification benchmarks, comparing the transformation weights ( $W_1$  and/or  $W_2$ ) learned in BGRL with random weights. **Red** indicates the best method, while underlined represents the second-best choice.

Method	Cora	CiteSeer	PubMed	Squirrel	Chameleon	Texas	Wisconsin	Cornell	Mean
BGRL	79.57 $\pm$ 0.90	68.88 $\pm$ 1.36	83.11 $\pm$ 0.40	<b>32.92 <math>\pm</math> 0.39</b>	46.02 $\pm$ 1.90	<b>85.74 <math>\pm</math> 3.11</b>	<u>72.75 <math>\pm</math> 2.00</u>	<u>80.49 <math>\pm</math> 1.64</u>	68.69
Randomize $W_1$	81.02 $\pm$ 0.64	<b>71.56 <math>\pm</math> 1.30</b>	83.11 $\pm$ 0.40	30.48 $\pm$ 0.70	<u>46.26 <math>\pm</math> 1.27</u>	85.25 $\pm$ 1.97	<b>85.63 <math>\pm</math> 3.00</b>	<b>80.98 <math>\pm</math> 1.97</b>	<b>70.54</b>
Randomize $W_2$	<b>82.97 <math>\pm</math> 1.05</b>	70.22 $\pm$ 1.02	<u>83.29 <math>\pm</math> 0.38</u>	<u>32.42 <math>\pm</math> 0.79</u>	<b>46.76 <math>\pm</math> 1.29</b>	85.41 $\pm$ 3.11	72.38 $\pm$ 2.00	<u>80.49 <math>\pm</math> 1.80</u>	<u>69.24</u>
Randomize both	<u>81.86 <math>\pm</math> 0.61</u>	<u>71.05 <math>\pm</math> 1.06</u>	<b>83.41 <math>\pm</math> 0.41</b>	30.99 $\pm$ 0.51	46.13 $\pm$ 1.36	<u>85.57 <math>\pm</math> 1.97</u>	72.63 $\pm$ 1.50	<b>80.98 <math>\pm</math> 1.97</b>	69.08

## E EXPERIMENTS WITH A FIXED PUBLIC-SPLITTING.

In Section 4.2, we evaluate PROP and other graph self-supervised methods on the node classification task with a random splitting. To avoid the conclusion working on one specific split setting, we here evaluate the models on the public fixed splits following Zhu et al. (2021c); Zhang et al. (2021). In practice, we use the public splitting introduced in Pei et al. (2020) for most datasets. There is no available public splitting for Amazon-Photo and Amazon-Computers, so we randomly split the

dataset into 1/1/8 as the train/validation/test set, differing from the splitting in Section 4.2. Other experimental settings are kept the same. As shown in Table 14, on 6 in 10 benchmarks PROP performs the best among baselines and exceeds the runner-up ProGCL by 4.23% on average. The results verify the effectiveness of PROP in different data-splitting cases.

Table 14: Test accuracy (%) of PROP and other graph self-supervised methods on node classification benchmarks with the public splitting. **Red** indicates the best method, while underlined represents the second-best choice.

Method	Cora	CiteSeer	PubMed	Photo	Computers	Squirrel	Chameleon	Texas	Wisconsin	Cornell	Mean
DeepWalk	80.87 $\pm$ 1.07	63.14 $\pm$ 1.05	81.55 $\pm$ 0.27	<u>84.66 <math>\pm</math> 0.40</u>	89.59 $\pm$ 0.18	43.32 $\pm$ 0.79	60.81 $\pm$ 1.27	53.44 $\pm$ 5.09	43.63 $\pm$ 4.25	44.59 $\pm$ 2.95	64.56
Node2Vec	84.27 $\pm$ 0.70	66.04 $\pm$ 1.83	81.33 $\pm$ 0.36	83.92 $\pm$ 0.31	89.31 $\pm$ 0.20	38.41 $\pm$ 1.19	59.50 $\pm$ 2.30	60.81 $\pm$ 1.89	55.10 $\pm$ 3.73	60.54 $\pm$ 3.24	67.92
GAE	85.96 $\pm$ 1.03	72.78 $\pm$ 1.11	85.06 $\pm$ 0.49	75.29 $\pm$ 0.53	89.50 $\pm$ 0.26	35.56 $\pm$ 1.27	56.51 $\pm$ 1.62	62.43 $\pm$ 4.86	61.18 $\pm$ 3.53	60.27 $\pm$ 3.51	68.45
VGAE	86.20 $\pm$ 0.76	73.26 $\pm$ 0.65	85.19 $\pm$ 0.43	72.17 $\pm$ 0.33	86.90 $\pm$ 0.38	42.38 $\pm$ 1.13	60.29 $\pm$ 1.05	63.78 $\pm$ 3.51	59.61 $\pm$ 2.75	60.54 $\pm$ 2.16	69.03
GRACE	84.10 $\pm$ 1.01	70.41 $\pm$ 0.92	84.79 $\pm$ 0.38	78.51 $\pm$ 0.44	87.80 $\pm$ 0.41	39.65 $\pm$ 0.87	55.83 $\pm$ 1.05	64.59 $\pm$ 4.59	58.82 $\pm$ 4.91	60.81 $\pm$ 2.16	68.53
DGI	<u>87.20 <math>\pm</math> 0.99</u>	72.50 $\pm$ 1.49	82.55 $\pm$ 0.38	71.35 $\pm$ 0.57	80.43 $\pm$ 0.63	36.61 $\pm$ 1.05	52.02 $\pm$ 1.32	<u>70.54 <math>\pm</math> 2.97</u>	63.53 $\pm$ 3.92	61.62 $\pm$ 2.16	67.84
MVGRL	83.44 $\pm$ 0.72	71.61 $\pm$ 0.73	82.48 $\pm$ 0.30	80.96 $\pm$ 0.67	86.87 $\pm$ 0.41	31.48 $\pm$ 0.83	58.77 $\pm$ 1.45	68.38 $\pm$ 2.98	62.94 $\pm$ 3.53	61.62 $\pm$ 2.16	68.86
CCA-SSG	<b>87.71 <math>\pm</math> 0.75</b>	<b>75.42 <math>\pm</math> 0.80</b>	85.55 $\pm$ 0.40	78.96 $\pm$ 0.33	<b>90.91 <math>\pm</math> 0.38</b>	40.16 $\pm$ 0.74	54.98 $\pm$ 1.18	68.65 $\pm$ 3.78	<u>64.12 <math>\pm</math> 4.31</u>	61.89 $\pm$ 2.43	70.84
BGRL	85.77 $\pm$ 0.89	72.66 $\pm$ 1.54	84.63 $\pm$ 0.49	74.43 $\pm$ 0.91	85.50 $\pm$ 0.59	37.20 $\pm$ 1.07	53.82 $\pm$ 1.67	67.03 $\pm$ 2.70	60.59 $\pm$ 3.14	60.81 $\pm$ 2.43	68.24
GCA	86.60 $\pm$ 0.79	<u>74.71 <math>\pm</math> 1.18</u>	<u>86.44 <math>\pm</math> 0.34</u>	75.63 $\pm$ 0.46	88.77 $\pm$ 0.54	41.33 $\pm$ 0.88	59.28 $\pm$ 1.54	69.46 $\pm$ 2.97	62.94 $\pm$ 2.75	<u>61.89 <math>\pm</math> 2.16</u>	70.71
ProGCL	85.45 $\pm$ 0.85	73.61 $\pm$ 1.10	<b>86.86 <math>\pm</math> 0.41</b>	81.64 $\pm$ 0.70	89.91 $\pm$ 0.31	<u>50.23 <math>\pm</math> 0.86</u>	<u>67.81 <math>\pm</math> 1.47</u>	69.46 $\pm$ 2.97	62.75 $\pm$ 2.75	61.35 $\pm$ 1.35	<u>72.91</u>
PROP	84.57 $\pm$ 0.82	74.55 $\pm$ 1.09	84.65 $\pm$ 0.24	<b>84.78 <math>\pm</math> 0.38</b>	<u>90.83 <math>\pm</math> 0.34</u>	<b>57.20 <math>\pm</math> 1.41</b>	<b>68.71 <math>\pm</math> 1.18</b>	<b>71.35 <math>\pm</math> 4.60</b>	<b>79.61 <math>\pm</math> 3.14</b>	<b>75.14 <math>\pm</math> 3.78</b>	<b>77.14</b>

## F FLIP EXPERIMENTS IN SECTION 5.2

In this flip experiment, we first train GRACE with ChebNetII as the encoder and save the learned transformation weights  $\mathbf{W}_{\text{CL}}$  and propagation coefficients  $\theta_{\text{CL}}$ . Then we train ChebNetII in the supervised setting with the propagation coefficients fixed with  $\theta_{\text{CL}}$ , or the transformation weights fixed with  $\mathbf{W}_{\text{CL}}$ . As shown in Table 15, despite using the propagation coefficients learned by GCL, the model still achieves satisfying performances compared to the original supervised model. However, after replacing the transformation weights, the performance deteriorates largely. The results further confirm our conclusion in Section 5.2

Table 15: Test accuracy (%) of node classification benchmarks. We freeze the propagation coefficients with optimal  $\theta_{\text{CL}}$  (or the transformation weights with  $\mathbf{W}_{\text{CL}}$ ), and *learn* the transformation weights (or propagation coefficients) in the supervised setting.  $\mathbf{1}$  denotes an all-one vector. **Red** indicates the best, while underlined represents the second-best choice.

Method	$\theta$	$\mathbf{W}$	Cora	CiteSeer	PubMed	Squirrel	Chameleon	Texas	Wisconsin	Cornell	Mean
SL	<i>Learn</i>	<i>Learn</i>	<b>88.39 <math>\pm</math> 0.74</b>	<b>79.67 <math>\pm</math> 0.72</b>	<b>87.11 <math>\pm</math> 0.25</b>	<b>49.34 <math>\pm</math> 1.09</b>	<b>69.52 <math>\pm</math> 0.96</b>	<b>89.67 <math>\pm</math> 2.13</b>	<b>91.25 <math>\pm</math> 2.75</b>	<b>88.36 <math>\pm</math> 3.11</b>	<b>80.41</b>
CL	$\theta_{\text{CL}}$	$\mathbf{W}_{\text{CL}}$	83.42 $\pm$ 0.92	74.79 $\pm$ 0.57	84.92 $\pm$ 0.26	37.90 $\pm$ 0.79	55.67 $\pm$ 0.96	77.87 $\pm$ 2.79	86.38 $\pm$ 3.63	75.74 $\pm$ 3.61	72.09
Fix-transformation	<i>Learn</i>	$\mathbf{W}_{\text{CL}}$	76.62 $\pm$ 2.12	76.25 $\pm$ 0.64	83.32 $\pm$ 0.46	36.56 $\pm$ 0.61	52.41 $\pm$ 2.06	60.16 $\pm$ 6.39	75.25 $\pm$ 4.38	59.51 $\pm$ 5.08	65.01
Fix-propagation	$\theta_{\text{CL}}$	<i>Learn</i>	<u>87.06 <math>\pm</math> 0.53</u>	<u>79.55 <math>\pm</math> 0.74</u>	<u>85.76 <math>\pm</math> 0.23</u>	<u>41.44 <math>\pm</math> 1.06</u>	<u>64.44 <math>\pm</math> 0.74</u>	<u>87.38 <math>\pm</math> 2.95</u>	<u>90.63 <math>\pm</math> 3.00</u>	<u>84.26 <math>\pm</math> 2.62</u>	<u>77.57</u>
All-one baseline	$\mathbf{1}$	<i>Learn</i>	71.74 $\pm$ 3.22	75.92 $\pm$ 0.61	79.38 $\pm$ 0.47	33.27 $\pm$ 0.61	42.32 $\pm$ 0.90	55.41 $\pm$ 4.43	74.13 $\pm$ 4.13	60.82 $\pm$ 6.56	61.65

## G AGGREGATION STEP IN PROP

In this section, we present the accuracies of PROP with different propagation steps. We find the best step choice varies among datasets, but a shallow propagation is enough in most cases. As shown in Figure 2, only one-step propagation performs best in datasets including Cora, CiteSeer, Chameleon, Squirrel, Computers, and Photo. For Texas, Wisconsin, Cornell, Actor, and CS, the raw features, (*i.e.*, zero propagation step) are enough. Moreover, when the performance achieves the best, raising the propagation step will cause a degradation.

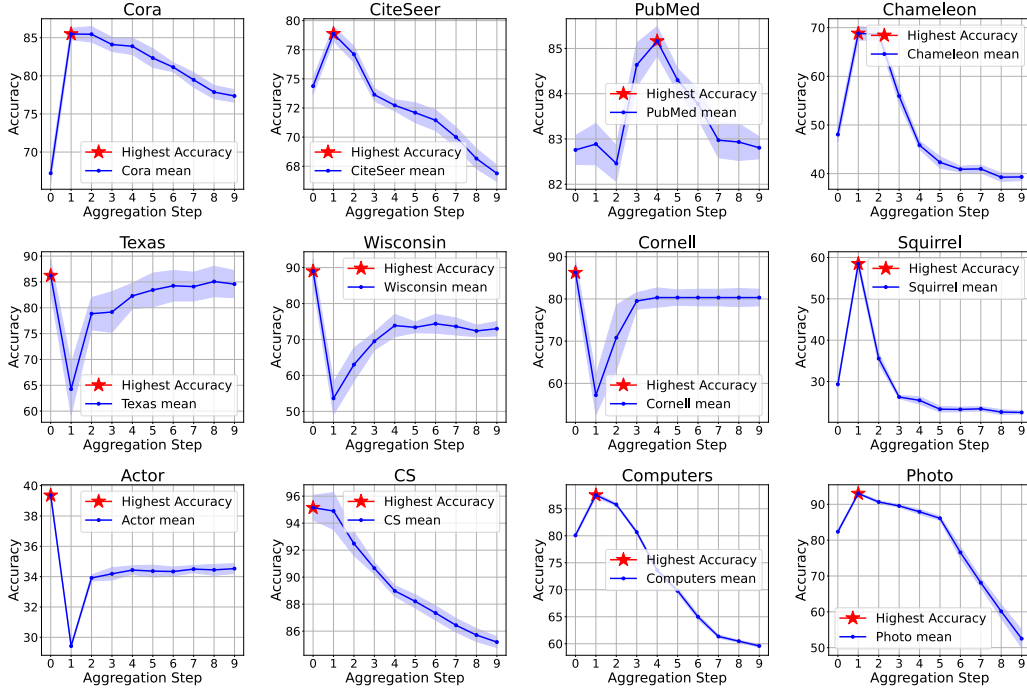


Figure 2: Accuracy (%) of PROP with different propagation steps. We mark the best step choice with a red star. Experiments are conducted ten times and the shadow denotes the derivation.

## H TRIALS ON LEARNING EFFECTIVE TRANSFORMATION WEIGHTS IN GCL

According to the analysis in Section 5.1, GCL learns uninformative weights that are excessively smoothing. Here we try three ways to solve this problem: 1) enforcing the sparsity of weights with  $l_1$  normalization; 2) using whitening methods (Bell & Sejnowski, 1997; Kessy et al., 2018); 3) using normalization methods (Huang et al., 2018; Hua et al., 2021; Guo et al., 2023a).

**$l_1$  regularization.** As a typical technique, the  $l_1$  regularization encourages sparsity by driving some weights to zero and retaining the most relevant features. In practice, we add a penalty proportional to the sum of the absolute values of the encoder parameters to the contrastive loss, *i.e.*,  $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{CL}} + \lambda \sum_i |w_i|$ , where  $\mathcal{L}_{\text{CL}}$  is the contrastive loss,  $\lambda$  is the regularization strength, and the  $w_i$  is the parameters of the encoder. We conduct experiments on ChebNetII with the  $l_1$  regularized GRACE training objective, varying the regularization strength  $\lambda$  in  $[1 \times 10^{-4}, 1 \times 10^{-5}, 1 \times 10^{-6}]$ . As shown in Table 16, the  $l_1$  regularization improves performance over the original GRACE on the Squirrel, Chameleon, Texas, Wisconsin, and Cornell datasets, though it still lags behind PROP, except on Wisconsin. However, for Cora, Citeseer, and PubMed,  $l_1$  regularization negatively impacts performance.

Table 16: Test accuracy (%) of node classification benchmarks. We train ChebNetII using the  $l_1$  regularized GRACE objective.  $\lambda$  denotes the regularization strength. Red indicates the best, while underlined represents the second-best choice.

	Cora	CiteSeer	PubMed	Squirrel	Chameleon	Texas	Wisconsin	Cornell
PROP	<b>85.48 <math>\pm</math> 0.76</b>	<b>78.87 <math>\pm</math> 0.63</b>	82.89 $\pm$ 0.48	<b>58.48 <math>\pm</math> 1.03</b>	<b>68.82 <math>\pm</math> 1.42</b>	<b>86.23 <math>\pm</math> 3.11</b>	89.00 $\pm$ 3.25	<b>86.23 <math>\pm</math> 3.11</b>
$\lambda=0$ (GRACE)	83.42 $\pm$ 0.92	<u>74.79 <math>\pm</math> 0.57</u>	<b>84.92 <math>\pm</math> 0.26</b>	37.90 $\pm$ 0.79	55.67 $\pm$ 0.96	77.87 $\pm$ 2.79	86.38 $\pm$ 3.63	75.74 $\pm$ 3.61
$\lambda=1e-4$	53.71 $\pm$ 1.10	26.97 $\pm$ 0.50	81.20 $\pm$ 0.21	33.07 $\pm$ 0.89	48.60 $\pm$ 1.42	<u>80.98 <math>\pm</math> 2.30</u>	70.00 $\pm$ 1.88	<u>82.79 <math>\pm</math> 2.46</u>
$\lambda=1e-5$	78.87 $\pm$ 1.17	73.29 $\pm$ 0.63	<u>84.17 <math>\pm</math> 0.23</u>	37.46 $\pm$ 0.89	56.37 $\pm$ 1.01	56.56 $\pm$ 1.97	<b>91.88 <math>\pm</math> 2.25</b>	81.80 $\pm$ 2.30
$\lambda=1e-6$	77.75 $\pm$ 0.80	73.90 $\pm$ 0.74	84.16 $\pm$ 0.21	<u>38.27 <math>\pm</math> 1.02</u>	<u>56.91 <math>\pm</math> 1.09</u>	52.79 $\pm$ 4.76	86.88 $\pm$ 2.88	74.26 $\pm$ 7.38



**Whitening methods.** Whitening methods are used to decorrelate and normalize data. By making dimensions mutually independent, whitening methods implicitly solve the representation collapse problem. Here we consider the typical Zero-phase Component Analysis (ZCA) whitening (Kessy et al., 2018), which transforms the input data such that it has zero mean and identity covariance matrix, while also preserving data structure as much as possible. It is computed by multiplying the data by the inverse square root of its covariance matrix, *i.e.*,  $\hat{x} = \mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}^\top \mathbf{x}$ , where  $\mathbf{V}$  is the matrix of eigenvectors and  $\mathbf{\Lambda}$  is the diagonal matrix of eigenvalues of the covariance matrix of  $\mathbf{x}$ . We conduct experiments under the GRACE framework with a ZCA whitening layer added to the encoder ChebNetII. As shown in Table 17, the whitening improves performance over the original GRACE on the PubMed and Chameleon datasets but drastically deteriorates most of the other datasets.

Table 17: Test accuracy (%) of node classification benchmarks. We train ChebNetII using GRACE with the ZCA whitening. **Red** indicates the best, while underlined represents the second-best choice.

	Cora	CiteSeer	PubMed	Squirrel	Chameleon	Texas	Wisconsin	Cornell
PROP	<b>85.48 ± 0.76</b>	<b>78.87 ± 0.63</b>	82.89 ± 0.48	<b>58.48 ± 1.03</b>	<b>68.82 ± 1.42</b>	<b>86.23 ± 3.11</b>	<u>89.00 ± 3.25</u>	<b>86.23 ± 3.11</b>
GRACE	83.42 ± 0.92	<u>74.79 ± 0.57</u>	<u>84.92 ± 0.26</u>	<u>37.90 ± 0.79</u>	55.67 ± 0.96	<u>77.87 ± 2.79</u>	<u>86.38 ± 3.63</u>	<u>75.74 ± 3.61</u>
GRACE+ZCA	79.29 ± 1.71	47.29 ± 0.70	<b>85.76 ± 0.29</b>	36.72 ± 0.91	<u>58.60 ± 1.07</u>	43.77 ± 8.36	27.38 ± 3.63	38.52 ± 6.23

**Normalization methods.** For normalization methods, we consider the widely used Batch Normalization (BN) (Ioffe, 2015), and the recently proposed Decorrelate ContraNorm (DCN) (Guo et al., 2023a). Batch normalization scales and shifts the mini-batch of data to have a mean of zero and a standard deviation of one, *i.e.*,  $\hat{x} = (\mathbf{x} - \boldsymbol{\mu}_B) / \sqrt{\boldsymbol{\sigma}_B^2 + \epsilon}$ , where  $\boldsymbol{\mu}_B$  and  $\boldsymbol{\sigma}_B^2$  are the mean and variance of the mini-batch  $B$ , and  $\epsilon$  is a small constant for numerical stability. DCN scatters representations in the embedding space and leads to a more uniform distribution. The formulation of GCN is  $\hat{x} = \mathbf{x} - s \times \mathbf{x} \times \text{softmax}(\mathbf{x}^\top \mathbf{x})$ , where  $s$  is the scale factor. We conduct experiments under the GRACE framework with a BN or DCN layer added to the encoder ChebNetII. As shown in Table 18, BN and DCN both fail to bring substantial improvement over the original GRACE.

Table 18: Test accuracy (%) of node classification benchmarks. We train ChebNetII using GRACE with BN or DCN normalization.  $s$  denotes the scale factor in DCN. **Red** indicates the best, while underlined represents the second-best choice.

	Cora	CiteSeer	PubMed	Squirrel	Chameleon	Texas	Wisconsin	Cornell
PROP	<b>85.48 ± 0.76</b>	<b>78.87 ± 0.63</b>	82.89 ± 0.48	<b>58.48 ± 1.03</b>	<b>68.82 ± 1.42</b>	<b>86.23 ± 3.11</b>	<u>89.00 ± 3.25</u>	<b>86.23 ± 3.11</b>
GRACE	83.42 ± 0.92	<u>74.79 ± 0.57</u>	<u>84.92 ± 0.26</u>	37.90 ± 0.79	55.67 ± 0.96	<u>77.87 ± 2.79</u>	86.38 ± 3.63	75.74 ± 3.61
GRACE + BN	82.25 ± 1.00	72.78 ± 1.00	<b>85.10 ± 0.24</b>	<u>39.56 ± 0.47</u>	54.77 ± 0.74	76.07 ± 2.95	72.63 ± 4.75	75.90 ± 2.79
GRACE + DCN ( $s=0.5$ )	79.79 ± 0.99	73.86 ± 0.86	84.00 ± 0.37	38.17 ± 0.95	56.19 ± 1.03	71.15 ± 2.13	83.25 ± 2.50	71.64 ± 4.59
GRACE + DCN ( $s=1.0$ )	75.19 ± 1.08	74.91 ± 0.63	83.06 ± 0.22	38.28 ± 1.12	57.35 ± 0.98	74.26 ± 1.64	<b>90.50 ± 1.50</b>	<u>76.72 ± 3.11</u>
GRACE + DCN ( $s=5.0$ )	74.40 ± 1.15	74.46 ± 0.63	79.41 ± 0.35	38.01 ± 0.79	<u>58.97 ± 1.33</u>	72.95 ± 3.44	83.25 ± 2.75	73.44 ± 3.44

In summary, these techniques offer limited effectiveness for GCL when used with polynomial GNNs. We think the possible reason is that the learning of transformation weights needs a high-quality supervision signal. Although these methods help prevent representation collapse, they do not carry extra information. Therefore, GCL still fails to learn good transformation weights.

## I HYPERPARAMETER SENSITIVITY ANALYSIS

In this section, we conduct the hyperparameter sensitivity analysis comparing PROPGCL and the corresponding backbone GCL methods. We vary the range of hyperparameters and evaluate the downstream performance. Here, we choose two hyperparameters in the model architecture, the hidden dimension and the propagation step. We consider the DGI backbone with the Chebyshev basis. As shown in Figure 3 and Figure 4, the performance of DGI with ChebNetII is highly influenced by disturbing hyperparameters. For example, on Cora, decreasing the hidden dimension from 256 to 128 causes nearly 40% accuracy degradation. In comparison, the performances of PROP-DGI show low variance under different hyperparameter combinations, and a sharp decline is only observed when using small neural networks.

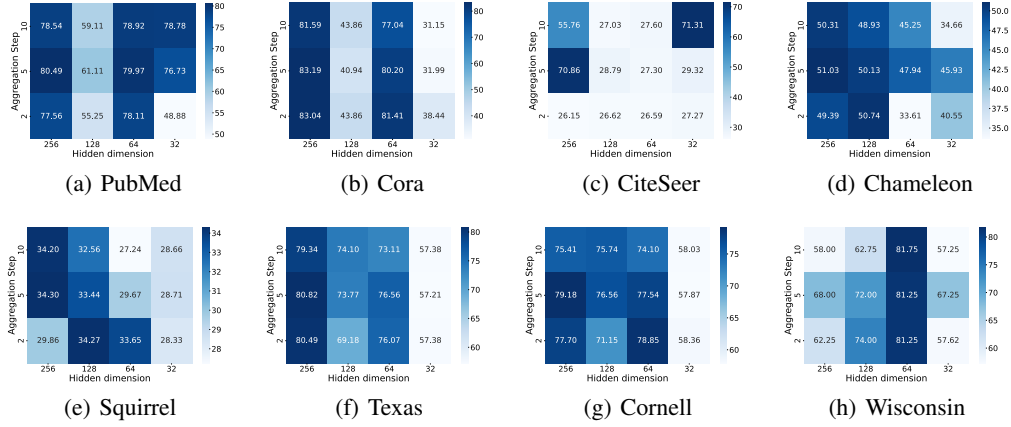


Figure 3: Hyperparameter sensitivity analysis on the hidden dimension and propagation step. Experiments are conducted on DGI with ChebNetII as the encoder.

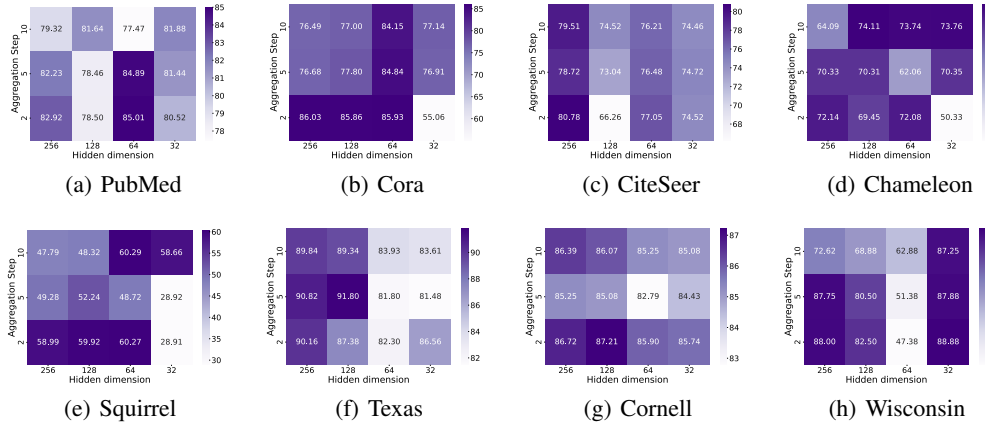


Figure 4: Hyperparameter sensitivity analysis on the hidden dimension and propagation step. Experiments are conducted on PROP-DGI with the Chebyshev basis.

## J DETAILS ABOUT POLYNOMIAL GNNs

In this section, we introduce polynomial GNNs from the spectral perspective. Developed from graph signal processing, *graph convolution* means transforming the graph signals to the Fourier domain and then back to the vertex domain after suitable filtering, *i.e.*,  $\mathbf{H} = \mathbf{U}g_\theta(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{X}$ , where  $g_\theta$  is the filter,  $\mathbf{U}$  is the matrix of eigenvectors of graph Laplacian  $\mathbf{L}$ ,  $\mathbf{\Lambda}$  is the diagonal matrix of eigenvalues. The problem arises when the parameters in  $g_\theta(\mathbf{\Lambda})$  are entirely unconstrained, leading to a lack of spatial localization in the convolution and a high time complexity due to eigenvalue decomposition.

These issues can be overcome with the use of a polynomial filter  $g_\theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k \mathbf{\Lambda}^k$ , where the parameter  $\theta \in \mathbb{R}^K$  is a vector of polynomial coefficients. Therefore, the graph convolution can be reformulated as  $\mathbf{H} = (\sum_{k=0}^{K-1} \theta_k \mathbf{L}^k) \mathbf{X}$ . We call GNNs using the polynomial approximated filters as *polynomial GNNs*. As one of the pioneer works, ChebNet (Defferrard et al., 2016) use Chebyshev polynomial parametrization to localize filters as  $g_\theta(\mathbf{\Lambda}) = \sum_{k=0}^K \theta_k T_k(\tilde{\mathbf{\Lambda}})$ , where  $\tilde{\mathbf{\Lambda}} = 2\mathbf{\Lambda}/\lambda_{\max} - \mathbf{I}$ ,  $\theta$  is the Chebyshev coefficients, and  $T_k(\tilde{\mathbf{\Lambda}})$  is the Chebyshev polynomial of order  $k$  recursively calculated by  $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$  with  $T_0(x) = 1$  and  $T_1(x) = x$ .

In section ??, we consider three popular polynomial GNN variants. GPRGNN (Chien et al., 2021) uses the monomial basis functions evaluated at  $\hat{\mathbf{A}}$ , *i.e.*,  $g_\theta(\mathbf{\Lambda}) = \sum_{k=0}^{K-1} \theta_k (\mathbf{I} - \hat{\mathbf{L}})^k$  with  $\theta$  as learnable coefficients. BernNet (He et al., 2021) uses the Bernstein polynomial approximation,

i.e.,  $g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \frac{1}{2^k} \binom{K}{k} (2\mathbf{I} - \mathbf{L})^{K-k} \mathbf{L}^k$  with  $\theta$  as learnable coefficients. ChebNetII (He et al., 2022) enhances the original Chebyshev polynomial approximation by Chebyshev interpolation, formulated as  $g_\theta(\Lambda) = \frac{2}{K+1} \sum_{k=0}^K \sum_{j=0}^K \theta_j T_k(x_j) T_k(\hat{\mathbf{L}})$ , where  $x_j = \cos((j+1/2)\pi/(K+1))$  are the Chebyshev nodes of  $T_{K+1}$ , and  $\theta$  are learnable coefficients.

## K CHARACTERIZATION OF LEARNED PROPAGATION COEFFICIENTS

In section 5.2, we find after replacing the transformation weights with supervised ones, the model trained in GCL performs as well as in a supervised manner. To show that given the transformation weights, GCL can learn effective propagation coefficients. We compare the propagation coefficients learned by SL, GCL, and the fix-transformation GCL. As shown in Figure 5, compared with CL, the propagation coefficients learned by fix-transformation GCL are closer to those in SL. Notably, the best propagation coefficients for one dataset may not be unique. Therefore, differing from the SL coefficients does not necessarily indicate poor quality, and the results can not prove that GCL learns bad propagation coefficients. However, it demonstrates that GCL can learn effective propagation coefficients fitting the given transformation weights.

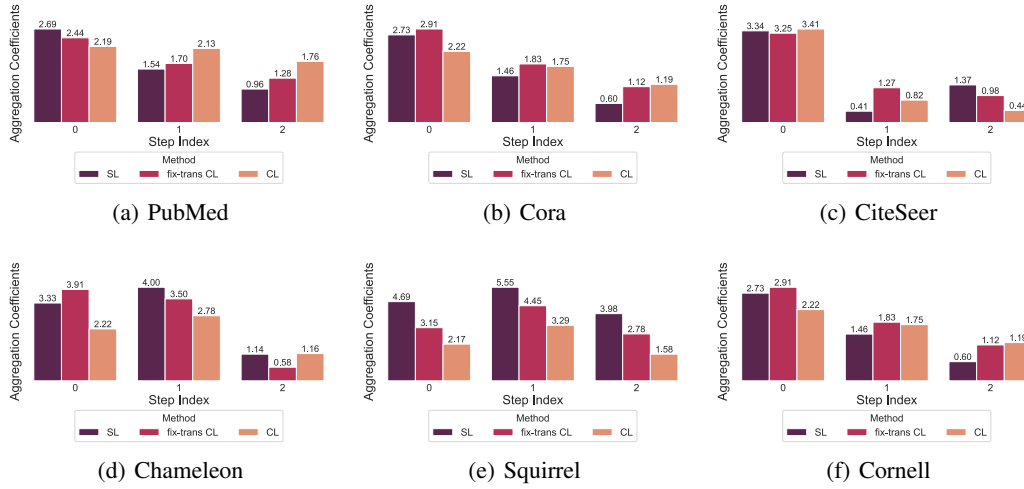


Figure 5: Propagation coefficients of the supervised learning (SL), the contrastive learning (CL), and the fix-transformation contrastive learning (fix-trans CL) introduced in Section 5.2. We show the first three propagation coefficients for the space limit.

## L CHARACTERIZATION OF LEARNED TRANSFORMATION WEIGHTS

In Section 5.1, we demonstrated the transformation weights learned by GCL and SL on the Cora dataset. Here, we extend these findings by presenting comprehensive results across various datasets. As depicted in Figure 6, the weights learned by GCL exhibit a smoother heatmap compared to those learned by SL. Furthermore, as shown in Figure 7, the weights learned by SL display diverse, data-dependent distributions, while those learned by CL consistently follow a Gaussian-like distribution. These results provide further evidence that GCL struggles to learn effective transformation weights.

## M EFFICIENCY ANALYSIS

PROPGCL is more efficient than the original baselines in time and memory consumption as shown in Table 19 and Table 20. Remarkably, PRO-GRACE saves 84.29% training time per epoch for the original GRACE with Chebyshev basis on Coauthor CS. For memory consumption, PROP-GRACE consumes over 99% less memory in the encoder for different benchmarks than the original baseline. The boost of time and memory efficiency of PROPGCL is attributed to the exclusion of transformation weights computation in self-supervised training.



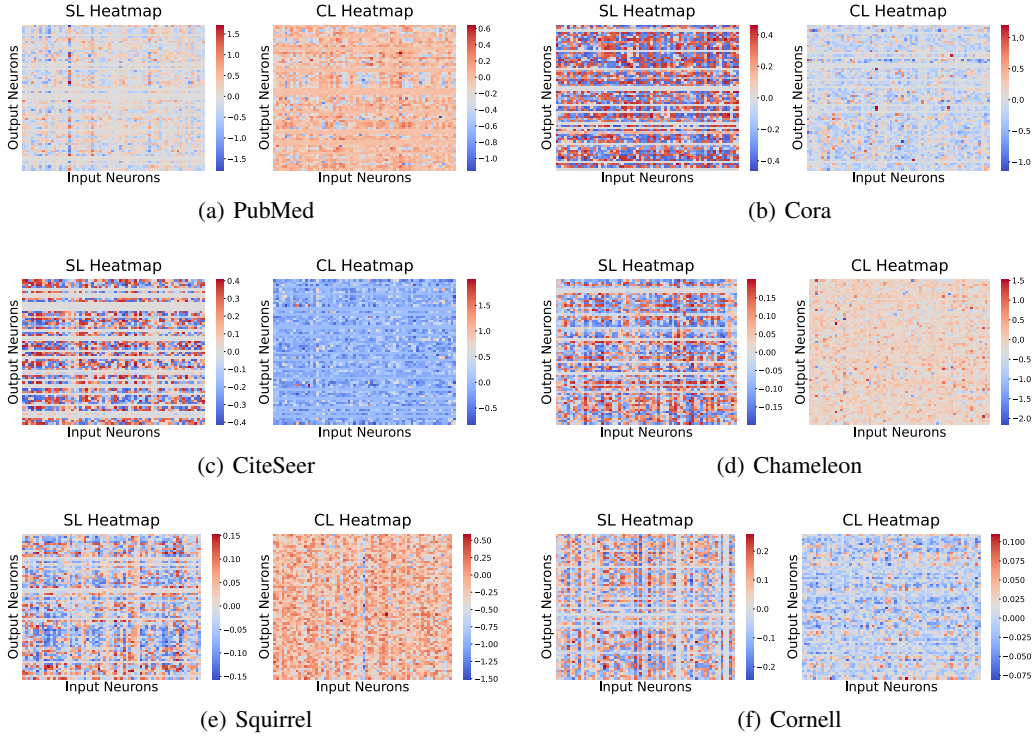


Figure 6: Heatmap of the transformation weights learned by GCL and SL.

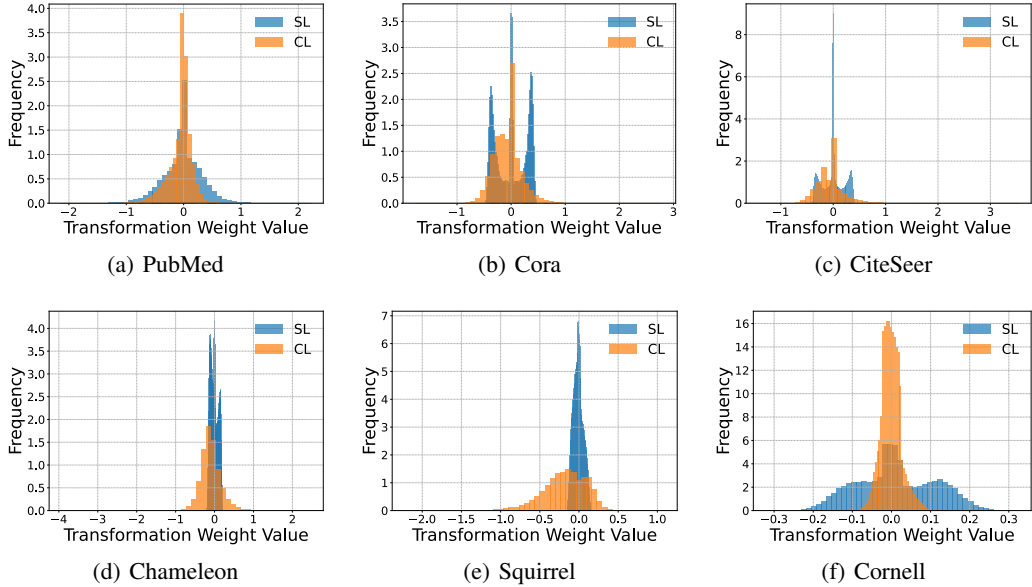


Figure 7: Distribution of the transformation weights learned by GCL and SL.

## N ANALYSIS ON BASIS POLYNOMIAL FUNCTIONS

Polynomial GNNs variants mainly differ in the polynomial basis function choices, *e.g.*, the monomial basis in GPRGNN (Chien et al., 2021), the Bernstein basis in BernNet (He et al., 2021), and the

Table 19: Comparison of training time per epoch in seconds between polynomial GNNs and its corresponding -PROP version in the GRACE framework. *Improvement* refers to the percentage increase in speed of the -PROP version compared to the baseline, *i.e.*,  $(t_{\text{GRACE}} - t_{\text{PROP-GRACE}})/t_{\text{GRACE}}$ . Experiments are all conducted on a single 24GB NVIDIA GeForce RTX 3090, except those denoted with \* on 48GB Nvidia A40 for out-of-memory.

Basis	Method	Cora	CiteSeer	PubMed	Photo	Computers	CS	Squirrel	Chameleon	Actor
Chebyshev	GRACE	0.1611	0.1939	0.2795	0.2872	0.4639	1.5111*	0.7004	0.2295	0.2872
	PROP-GRACE	0.1409	0.1478	0.2650	0.2400	0.3626	0.2374*	0.2581	0.1450	0.2073
	<i>Improvement</i>	12.54%	23.79%	5.18%	16.44%	21.84%	84.29%	63.15%	36.82%	27.83%
Bernstein	GRACE	0.1515	0.2215	0.2513	0.4878	0.9293	6.7666*	1.8997	0.4079	0.2619
	PROP-GRACE	0.1226	0.1178	0.2334	0.3832	0.6968	0.6038*	0.5175	0.1653	0.1789
	<i>Improvement</i>	19.03%	46.79%	7.10%	21.45%	25.02%	91.08%	72.76%	59.47%	31.69%
Monomial	GRACE	0.1114	0.1023	0.1217	0.1606	0.2340	1.2487*	0.3714	0.1524	0.1202
	PROP-GRACE	0.1024	0.1224	0.1221	0.1428	0.1928	0.1927*	0.1650	0.1151	0.1109
	<i>Improvement</i>	8.06%	16.42%	0.31%	11.12%	17.61%	84.57%	55.56%	24.46%	7.74%

Table 20: Comparison of memory consumption of encoder in KBs between PROPGCL and the original baseline. We consider GRACE with the Chebyshev basis function here. *Improvement* refers to the percentage decrease in the memory consumption of the -PROP version compared to the baseline. *i.e.*,  $(m_{\text{GRACE}} - m_{\text{PROP-GRACE}})/m_{\text{GRACE}}$ .

Encoder	Cora	CiteSeer	PubMed	Photo	Computers	CS	Squirrel	Chameleon	Actor
GRACE	3894.04	8434.04	2028.04	2518.04	2562.04	2562.04	5206.04	5678.04	2892.04
PROP-GRACE	11.24	28.97	3.95	5.86	6.04	6.04	16.36	18.21	7.32
<i>Improvement</i>	99.71%	99.66%	99.81%	99.77%	99.76%	99.76%	99.69%	99.68%	99.75%

Chebyshev basis in ChebNetII (He et al., 2022). We introduce detailed basis function formulations in Appendix J.

In this section, we compare different basis polynomial functions used in PROPGCL. Here we consider the Chebyshev basis, Bernstein basis, and monomial basis. As shown in Table 21 and Table 22, the performance of PROPGCL is relatively robust in the choice of basis functions. For homophily benchmarks, PROP-GRACE with Chebyshev basis and the PROP-DGI with monomial basis achieve the best, surpassing the second slightly by 0.05% on average. For heterophily benchmarks, the best PROP-DGI with the Chebyshev basis achieves 73.71% on average, and the Bernstein basis ranks second. In general, the Chebyshev basis is preferred in PROPGCL.

Table 21: Test accuracy (%) of homophily node classification benchmarks, comparing different basis functions in PROPGCL. **Red** indicates the best method, while underlined represents the second-best.

Method	Basis	Cora	CiteSeer	PubMed	Photo	Computers	CS	Mean
PROP-GRACE	Chebyshev	<u>87.42 ± 0.95</u>	81.56 ± 0.83	86.19 ± 0.35	93.32 ± 0.31	88.12 ± 0.23	<u>95.95 ± 0.14</u>	<b>88.76</b>
	Bernstein	<b>87.52 ± 1.20</b>	<u>81.69 ± 0.86</u>	85.90 ± 0.25	93.42 ± 0.24	87.77 ± 0.22	<b>95.97 ± 0.13</b>	<u>88.71</u>
	monomial	87.34 ± 1.13	<b>81.86 ± 0.79</b>	<u>86.41 ± 0.23</u>	93.19 ± 0.26	86.85 ± 0.34	95.91 ± 0.15	88.59
PROP-DGI	Chebyshev	86.19 ± 1.05	80.78 ± 0.65	85.14 ± 0.22	92.78 ± 0.37	<b>89.81 ± 0.20</b>	95.82 ± 0.18	88.42
	Bernstein	86.49 ± 0.99	80.93 ± 0.72	85.80 ± 0.40	<u>93.53 ± 0.26</u>	<u>89.77 ± 0.25</u>	95.46 ± 0.16	88.66
	monomial	86.86 ± 1.02	<u>81.69 ± 0.86</u>	<b>86.56 ± 0.33</b>	<b>93.72 ± 0.25</b>	88.18 ± 0.34	95.57 ± 0.14	<b>88.76</b>

## O EXPERIMENTAL DETAILS

### O.1 BENCHMARKS

**Node classification benchmarks.** 1) *Citation Networks* (Sen et al., 2008; Namata et al., 2012). Cora, CiteSeer, and PubMed are three popular citation graph datasets. In these graphs, nodes represent

Table 22: Test accuracy (%) of heterophily node classification benchmarks, comparing different basis functions in PROPGCL. Red indicates the best method, while underlined represents the second-best.

Method	Basis	Squirrel	Chameleon	Actor	Texas	Wisconsin	Cornell	Mean
PROP-GRACE	Chebyshev	55.09 $\pm$ 0.81	71.73 $\pm$ 1.18	39.35 $\pm$ 0.81	89.84 $\pm$ 1.81	88.50 $\pm$ 3.63	86.72 $\pm$ 2.46	71.87
	Bernstein	48.51 $\pm$ 0.85	70.02 $\pm$ 0.88	39.33 $\pm$ 0.81	90.16 $\pm$ 1.31	<u>89.00 <math>\pm</math> 3.25</u>	<b>88.52 <math>\pm</math> 2.95</b>	70.92
	monomial	51.96 $\pm$ 0.69	69.28 $\pm$ 1.05	39.52 $\pm$ 0.89	84.43 $\pm$ 2.62	84.13 $\pm$ 4.50	88.20 $\pm$ 2.79	69.59
PROP-DGI	Chebyshev	<b>60.53 <math>\pm</math> 0.66</b>	<b>74.11 <math>\pm</math> 0.96</b>	<b>39.53 <math>\pm</math> 0.84</b>	91.80 $\pm$ 2.30	88.88 $\pm$ 2.50	87.38 $\pm$ 2.62	<b>73.71</b>
	Bernstein	53.08 $\pm$ 0.83	71.20 $\pm$ 0.81	39.48 $\pm$ 0.77	<u>92.46 <math>\pm</math> 1.48</u>	<b>91.63 <math>\pm</math> 3.00</b>	87.38 $\pm$ 2.63	<u>72.54</u>
	monomial	<u>56.65 <math>\pm</math> 0.77</u>	<u>72.12 <math>\pm</math> 0.72</u>	37.80 $\pm$ 0.57	<b>93.11 <math>\pm</math> 1.80</b>	83.63 $\pm$ 5.88	81.97 $\pm$ 2.95	70.88

papers and edges correspond to the citation relationship between two papers. Nodes are classified according to academic topics. 2) *Amazon Co-purchase Networks* (Shchur et al., 2018). Photo and Computers are collected by crawling Amazon websites. Goods are represented as nodes and the co-purchase relationships are denoted as edges. Node features are the bag-of-words representation of product reviews. Each node is labeled with the category of goods. 3) *Wikipedia Networks* (Rozemberczki et al., 2021). Squirrel and Chameleon are collected from the English Wikipedia, representing page-page networks on specific topics. Nodes represent articles and edges are mutual links between them. 4) *WebKB Networks* (Pei et al., 2020). In Texas, Wisconsin, and Cornell datasets, nodes represent web pages and edges represent hyperlinks between them. Node features are the bag-of-words representation of web pages. 5) *Actor Networks* Pei et al. (2020). Each node corresponds to an actor, and the edge between two nodes denotes co-occurrence on the same Wikipedia page. Node features correspond to some keywords on the Wikipedia pages. Statistics of datasets are shown in Table 23.

Table 23: Statistics of node classification benchmarks.  $\mathcal{H}(G)$  denotes the edge homophily ratio introduced in Zhu et al. (2020a).

Homo / Hetero	Category	Dataset	# Nodes	# Edges	# Features	# Classes	$\mathcal{H}(G)$
Homophily	Citation	Cora	2,708	5,278	1,433	7	0.81
		CiteSeer	3,327	4,552	3,703	6	0.74
		PubMed	19,717	44,338	500	3	0.80
	Co-purchase	Photo	7,650	119,081	745	8	0.83
		Computers	13,752	245,861	767	10	0.78
Heterophily	Wikipedia	Chameleon	2,277	36,101	2,325	6	0.23
		Squirrel	5,201	217,073	2,089	4	0.22
		Texas	183	279	1703	5	0.11
	WebKB	Wisconsin	251	466	1703	5	0.21
		Cornell	183	277	1703	5	0.30
	Film-actor	Actor	7,600	30,019	932	5	0.22

**Graph Classification benchmarks.** 1) *Molecules*. MUTAG (Debnath et al., 1991) is a dataset of nitroaromatic compounds and the goal is to predict their mutagenicity on Salmonella Typhimurium. NCI1 (Wale et al., 2008) is a dataset of chemical molecules that are annotated based on their activity against non-small cell lung cancer and ovarian cancer cell lines. 2) *Bioinformatics*. PROTEINS (Borgwardt et al., 2005) is a dataset of proteins that are classified as enzymes or non-enzymes. Nodes represent the amino acids and two nodes are connected by an edge if they are less than 6 Angstroms apart. DD (Dobson & Doig, 2003) consists of protein structures with nodes corresponding to amino acids and edges indicating that two amino acids are within a certain number of angstroms. 3) *Social Networks*. IMDB-BINARY and IMDB-MULTI (Yanardag & Vishwanathan, 2015) are movie collaboration datasets consisting of a network of 1,000 actors/actresses who played roles in movies in IMDB. In each graph, nodes represent actors/actresses; corresponding nodes are connected if they appear in the same movie. COLLAB (Yanardag & Vishwanathan, 2015) is derived from three public collaboration datasets representing scientific collaborations between authors. For all benchmarks, we use collections from TUDataset (Morris et al., 2020). Statistics of datasets are shown in Table 24.

Table 24: Statistics of graph classification benchmarks. We report average numbers of nodes, edges, and features across graphs in graph classification datasets.

Category	Dataset	#Graphs	# Nodes	# Edges	# Features	# Classes
Moleculars	MUTAG	188	17.9	39.6	7	2
	NCII	4110	29.87	32.30	37	2
Proteins	PROTEINS	1113	39.1	145.6	0	2
	DD	1178	284.32	715.66	89	2
Social Networks	IMDB-BINARY	1000	19.8	193.1	0	2
	IMDB-MULTI	1500	13.0	131.9	0	3
	COLLAB	5000	74.49	2457.78	0	3

## O.2 BASELINES

We categorize baselines for the **node classification task** into 1) traditional graph embedding algorithms DeepWalk (Perozzi et al., 2014) and Node2Vec (Grover & Leskovec, 2016); 2) graph autoencoders GAE (Kipf & Welling, 2016), VGAE (Kipf & Welling, 2016); 3) graph contrastive methods GRACE (Zhu et al., 2020b), DGI (Velickovic et al., 2019), GCA (Zhu et al., 2021c), MV-GRL (Hassani & Khasahmadi, 2020), ProGCL (Xia et al., 2022); 4) graph non-contrastive methods CCA-SSG (Zhang et al., 2021) and BGRL (Thakoor et al., 2022), 5) heterophily baselines compared in Section 6.2, PolyGCL (Chen et al., 2024), HGRL (Chen et al., 2022), GraphACL (Xiao et al., 2024), SP-GCL (Wang et al., 2023), DSSL (Xiao et al., 2022). The design details are as follows.

### 1) Traditional graph embeddings.

- **DeepWalk** (Perozzi et al., 2014). DeepWalk leverages truncated random walks to capture local network structures. The algorithm treats the random walks as sequences of nodes, akin to sentences in language models. It learns latent representations by applying skip-gram to maximize the co-occurrence probabilities of nodes appearing in these random walks.
- **Node2Vec** (Grover & Leskovec, 2016). Node2Vec is built on DeepWalk by introducing a flexible biased random walk strategy to explore network neighborhoods. The key innovation is balancing breadth-first sampling (BFS) and depth-first sampling (DFS). This allows Node2Vec to capture both homophily and structural equivalence, making the learned node embeddings more expressive.

### 2) Graph autoencoders.

- **GAE** (Kipf & Welling, 2016). GAE involves an encoder-decoder architecture, where the encoder is a GCN that transforms node features into latent embeddings by aggregating information from neighboring nodes. The embeddings are then used by the decoder, which typically applies a simple inner product operation to reconstruct the graph structure, such as predicting edges between nodes.
- **VGAE** (Kipf & Welling, 2016). VGAE extends GAE by introducing a probabilistic framework using a variational autoencoder (VAE) setup. It models latent variables with Gaussian distributions, enabling the generation of node embeddings that capture uncertainty. This design improves the model’s ability to capture complex structures in graphs, especially in tasks like link prediction.

### 3) Graph contrastive methods.

The mode of GCL has three mainstreams: local-to-local, global-to-global, and global-to-local (Zhu et al., 2021b). A classic example of local-to-local is GRACE (Zhu et al., 2020b), which generates two graph views by augmentations and the same nodes in augmented views are positive while all the other node pairs are negative. Global-to-global mode is often used with multiple graphs in the graph classification task, with GraphCL (You et al., 2020) as an early but influential trial. For the global-to-local perspective, positive pairs are taken as the global representation and nodes of augmented views, and negative pairs are the global representation and nodes of corrupted views. DGI (Velickovic et al., 2019) is a typical example.

- **GRACE** (Zhu et al., 2020b). GRACE generates two graph views by corruption and learns node representations by maximizing the agreement of node representations in these two views. To provide diverse node contexts for the contrastive objective, GRACE proposes a hybrid scheme for generating graph views on both structure and attribute levels.
- **GCA** (Zhu et al., 2021c). GCA proposes adaptive augmentation that incorporates various priors for topological and semantic aspects of the graph. On the topology level, GCA designs augmentation schemes based on node centrality measures, while on the node attribute level, GCA corrupts node features by adding more noise to unimportant node features.
- **DGI** (Velickovic et al., 2019). DGI relies on maximizing mutual information between patch representations and corresponding high-level summaries of graphs—both derived using established graph convolutional network architectures. The learned patch representations summarize subgraphs centered around nodes of interest, and can thus be reused for downstream node-wise learning tasks.
- **MVGRL** (Hassani & Khasahmadi, 2020). MVGRL introduces a self-supervised approach for learning node and graph-level representations by contrasting structural views of graphs. MVGRL shows that contrasting multi-scale encodings does not improve performance, and the best performance is achieved by contrasting encodings from first-order neighbors and graph diffusion.
- **ProGCL** (Xia et al., 2022). ProGCL observes limited benefits when adopting existing hard negative mining techniques of other domains in graph contrastive learning. ProGCL proposes an effective method to estimate the probability of a negative being true and devises two schemes to boost the performance of GCL.

#### 4) *Non-contrastive methods.*

- **CCA-SSG** (Zhang et al., 2021). CCA-SSG optimizes a novel feature-level objective that aligns features across different graph augmentations. It uses decorrelation to prevent degenerate solutions, allowing the model to learn invariant node representations. The model avoids a mutual information estimator or negative samples, which simplifies training and reduces computational complexity.
- **BGRL** (Thakoor et al., 2022). BGRL avoids the use of negative samples by predicting different augmentations of the input graph. BGRL relies on a bootstrapping mechanism, where one branch predicts the output of another branch that is not updated by gradient descent. This method eliminates the complexity of contrastive learning and negative sampling, making it more scalable.

#### 5) *Heterophily baselines.*

- **PolyGCL** (Chen et al., 2024). PolyGCL integrates spectral polynomial filters into graph contrastive learning, enabling it to handle both homophilic and heterophilic graphs. The method generates different spectral views using polynomials and incorporates high-pass information into the contrastive objective.
- **HGRL** (Chen et al., 2022). HGRL introduces self-supervised learning for heterophilic graphs by capturing distant neighbors and preserving original node features. It achieves this through carefully designed pretext tasks optimized via high-order mutual information, avoiding reliance on labels.
- **GraphACL** (Xiao et al., 2024). GraphACL focuses on an asymmetric view of neighboring nodes. The algorithm captures both one-hop local neighborhood information and two-hop monophily similarity, crucial for modeling heterophilic structures.
- **SP-GCL** (Wang et al., 2023). SP-GCL introduces a single-pass graph contrastive learning method without augmentations. It theoretically guarantees performance across both homophilic and heterophilic graphs by studying the concentration property of features obtained through neighborhood propagation.
- **DSSL** (Xiao et al., 2022). DSSL decouples neighborhood semantics in self-supervised learning for node representation. It introduces a latent variable model that decouples node and link generation, making it flexible to different graph structures. The method utilizes variational inference for scalable optimization, improving downstream performance without relying on homophily assumptions.



We categorize the baselines in the **graph classification task** into 1) graph kernel methods including GL (Shervashidze et al., 2009), WL (Shervashidze et al., 2011), and DGK (Yanardag & Vishwanathan, 2015), 2) traditional graph embedding methods including node2vec (Grover & Leskovec, 2016), sub2vec (Adhikari et al., 2018), and graph2vec (Narayanan et al., 2017), 3) contrastive learning methods including InfoGraph (Sun et al., 2020), GraphCL (You et al., 2020), MVGRL (Hassani & Khasahmadi, 2020), JOAOv2 (You et al., 2021), ADGCL (Suresh et al., 2021) as introduced in recent works. The design details are as follows.

#### 1) Graph kernel methods.

- **Graphlet Kernel (GL)** (Shervashidze et al., 2009). GL works by counting the number of small subgraphs (known as graphlets) of a fixed size that appear in each graph. The comparison of these counts across graphs allows the kernel to capture the local topological structures of the graphs, making it useful for tasks such as graph classification.
- **Weisfeiler-Lehman Sub-tree Kernel (WL)** (Shervashidze et al., 2011). WL extends the concept of graph kernels by applying the Weisfeiler-Lehman test of isomorphism on graphs. It involves iteratively relabeling the nodes of the graphs based on the labels of their neighbors and then using these relabelings to define a kernel, typically counting matching sub-trees.
- **Deep Graph Kernel (DGK)** (Yanardag & Vishwanathan, 2015). DGK combines deep learning techniques with graph kernels. It first learns a low-dimensional representation of the graphs through unsupervised learning (often using a form of graph embedding or autoencoders), then applies traditional kernel methods to these representations.

#### 2) Traditional graph embeddings.

- **Node2Vec** (Grover & Leskovec, 2016). Node2Vec is built on DeepWalk by introducing a flexible biased random walk strategy to explore network neighborhoods. The key innovation is balancing BFS and DFS. This allows Node2Vec to capture both homophily and structural equivalence, making the learned node embeddings more expressive.
- **Sub2Vec** (Adhikari et al., 2018). Inspired by the word2vec model, sub2vec learns vector representations for subgraphs in a graph. It treats each subgraph as a "word" and the entire graph as a "document" to learn embeddings that capture the structural and contextual properties of subgraphs.
- **Graph2Vec** (Narayanan et al., 2017). Similar to sub2vec, graph2vec is designed to learn embeddings for entire graphs. By treating each graph as a "document" and graph substructures as "words," graph2vec employs a document embedding approach to learn a fixed-size vector representation for each graph.

#### 3) Graph contrastive methods.

- **GraphCL** (You et al., 2020). GraphCL designs four types of graph augmentations to incorporate various priors and learns graph-level representations by maximizing the global representations of two views for a graph.
- **InfoGraph** (Sun et al., 2020). InfoGraph maximizes the mutual information between the graph-level representation and the representations of substructures of different scales (*e.g.*, nodes, edges, triangles). By doing so, the graph-level representations encode aspects of the data that are shared across different scales of substructures.
- **ADGCL** (Suresh et al., 2021). ADGCL proposes a novel principle, adversarial GCL, which enables GNNs to avoid capturing redundant information during training by optimizing adversarial graph augmentation strategies used in GCL.
- **JOAO** (You et al., 2021). JOAO proposes a unified bi-level optimization framework to automatically, adaptively, and dynamically select data augmentations when performing GraphCL on specific graph data. JOAO is instantiated as min-max optimization.

### O.3 SETTINGS

For the node classification task, following Zhu et al. (2020b); Velickovic et al. (2019); Hassani & Khasahmadi (2020), we use linear evaluation protocol, where the model is trained in an unsupervised manner and feeds the learned representation into a linear logistic regression classifier. In the evaluation



procedure, we randomly split each dataset with a training ratio of 0.8 and a test ratio of 0.1, and hyperparameters are fixed the same way for all the experiments. Each experiment is repeated ten times with mean and standard derivation of accuracy score.

For the graph classification task, we use Adam SGD optimizer with the learning rate selected in  $\{10^{-3}, 10^{-4}, 10^{-5}\}$  and the number of epochs in  $\{20, 100\}$ . For PROP, we only search the propagation step  $K$  in the range of  $[0, 1, 2, 3, 5, 10]$ . Following Sun et al. (2020); You et al. (2020), we feed the generated graph embeddings into a linear Support Vector Machine (SVM) classifier, and the parameters of the downstream classifier are independently tuned by cross-validation. The C parameter is tuned in  $\{10^{-3}, 10^{-2}, \dots, 10^2, 10^3\}$ . We report the mean 10-fold cross-validation accuracy with standard deviation. All experiments are conducted on a single 24GB NVIDIA GeForce RTX 3090.

#### O.4 HYPERPARAMETER

For all methods, we train the linear classifier for 2000 epochs with a learning rate of 0.01 and no weight decay. For hyperparameters of the model architecture and the unsupervised training procedure, we maintain consistency in the hyperparameter search space across methods as much as possible.

Specifically, for GRACE, we search the temperature  $\tau$  in  $[0.1, 0.5, 1.0]$ , the projector hidden dimension in  $[128, 256, 512]$ , the learning rate in  $[0.01, 0.001]$ , fix the patience as 50, and all augmentation rates as 0.2. For DGI, we search the learning rate in  $[0.01, 0.001]$ , the early-stopping patience in  $[50, 100]$ , and the hidden dimension in  $[128, 256, 512]$ . For CCA-SSG, we search the training epochs in  $[20, 50, 100]$ ,  $\lambda$  in  $[1e-3, 5e-4]$ , the hidden dimension in  $[128, 256, 512]$ , and fix all augmentation ratios as 0.2. For GCA, we search the temperature  $\tau$  in  $[0.1, 0.5, 1.0]$ , the projector hidden dimension in  $[128, 256, 512]$ , the drop scheme in  $[pr, degree, evc]$ , and fix the early-stopping patience as 50, the learning rate as 0.01, and all augmentation ratios as 0.2. For BGRL, we search the predictor hidden dimension in  $[128, 256, 512]$ , the learning rate in  $[1e-4, 1e-5]$ , the weight decay in  $[0, 1e-5]$ , fix the learning rate warmup epochs as 1000, the momentum moving as 0.99. For DeepWalk, we search the vector dimension in  $[128, 256, 512]$ , the context window size in  $[5, 10]$ , the walk number in  $[10, 20]$ , and the walk length in  $[40, 80]$ . For Node2Vec, we search the vector dimension in  $[128, 256, 512]$ , the walk number in  $[10, 20]$ , the probability  $p$  in  $[0.5, 1.0]$ ,  $q$  in  $[0.5, 1.0]$ , and fix the context window size as 10, and the walk length as 80. For MVGRL, we search the learning rate in  $[0.01, 0.001]$ , the early stopping patience in  $[50, 100]$ , and the hidden dimension in  $[128, 256, 512]$ . For GAE and VGAE, we search the learning rate in  $[0.01, 0.001]$ , the early stopping patience in  $[50, 100]$ , and the hidden dimension in  $[128, 256, 512]$ . For the heterophily baselines in 6.2, we use the optimal hyperparameter combinations provided in the original papers.

## P PROOF OF THEOREMS

### Q PROOF OF THEOREM 4.1

Here we present the proof of Theorem 4.1.

*Proof.* The gradient update of the Dirichlet energy objective (Equation 2) gives the following update rule of node features  $\mathbf{H}$ ,

$$\mathbf{H} - \alpha \frac{\partial \mathcal{L}(\mathbf{H})}{\partial \mathbf{H}} = \mathbf{H} - 2\alpha \hat{\mathbf{L}}\mathbf{H} = ((1 - 2\alpha)\mathbf{I} + 2\alpha \hat{\mathbf{A}})\mathbf{H}, \quad (7)$$

where the  $\alpha$  is the step size. When we choose the learning rate  $\alpha = 0.5$ , we recover the propagation operation in Equation 1, i.e.,  $\mathbf{H}_{\text{new}} = \hat{\mathbf{A}}\mathbf{H}$ .

For convergence analysis, we have

$$\begin{aligned} \mathcal{L}(\mathbf{H}^{(K)}) &= (\hat{\mathbf{A}}^K \mathbf{H}^{(0)})^\top \hat{\mathbf{L}} (\hat{\mathbf{A}}^K \mathbf{H}^{(0)}) \\ &= \mathbf{H}^{(0)\top} \hat{\mathbf{A}}^K \hat{\mathbf{L}} \hat{\mathbf{A}}^K \mathbf{H}^{(0)} \\ &= \mathbf{H}^{(0)\top} (\hat{\mathbf{A}}^{2K} - \hat{\mathbf{A}}^{2K+1}) \mathbf{H}^{(0)}. \end{aligned} \quad (8)$$

As is known, the range of eigenvalue of  $\hat{\mathbf{L}}$  is  $[0, 2]$ , therefore, the eigenvalues of  $\hat{\mathbf{A}}$  belong to  $[-1, 1]$ . The eigenvalue of  $\hat{\mathbf{L}}$  equals 2 if and only if the graph is bipartite. So for non-bipartite graphs, which

is often the case for complex graphs in real world, we have the eigenvalues of  $\hat{\mathbf{A}}$  belong to  $(-1, 1]$ . Then when  $K$  goes towards infinity, we have  $\lim_{K \rightarrow +\infty} \mathcal{L}(\mathbf{H}^{(K)}) = 0$ , which ends the proof.  $\square$

## R PROOF OF THEOREM 4.2

Here we present the proof of Theorem 4.2.

*Proof.* A key step is to notice that the alignment objective Equation 3 is closely relevant to the Dirichlet energy when  $f(x_i) = \mathbf{H}_i, \forall i \in [N]$  :

$$\mathcal{L}_{\text{align}}(f) = - \sum_{i,j} \mathbf{A}_{ij} [\mathbf{H}_i^\top \mathbf{H}_j] / (\sum_{i,j} \mathbf{A}_{ij}) = \mathbf{H}^\top \mathbf{A} \mathbf{H} / (\sum_{i,j} \mathbf{A}_{ij}) = \mathbf{H}^\top (\mathbf{I} - \mathbf{L}) \mathbf{H} / (\sum_{i,j} \mathbf{A}_{ij}). \quad (9)$$

It is easy to see that graph convolution converges to identical vectors, known as oversmoothing. Therefore, we have  $\forall i, j, (\mathbf{H}_\infty)_i = (\mathbf{H}_\infty)_j$ . Therefore,

$$\lim_{k \rightarrow \infty} \mathcal{L}_{\text{align}}(f_k) = \mathbf{H}_\infty^\top \mathbf{A} \mathbf{H}_\infty / (\sum_{i,j} \mathbf{A}_{ij}) = (\sum_{i,j} \mathbf{A}_{ij}) / (\sum_{i,j} \mathbf{A}_{ij}) = -1,$$

which concludes the proof.  $\square$