A Proof of Theorem 2

We use i to indicate the index of $\lambda^{-1} = \max_{i \in N} |\lambda_{i+1} - \lambda_i|^{-1}$ and perturb graph Laplacian $\mathbf{L}$ by perturbing the eigenvectors. Specifically, we set

$$\begin{aligned} \mathbf{u}'_i =& \sqrt{1-\epsilon^2}\mathbf{u}_i + \epsilon \mathbf{u}_{i+1}, \\ \mathbf{u}'_{i+1} =& -\epsilon \mathbf{u}_i + \sqrt{1-\epsilon^2}\mathbf{u}_{i+1}. \end{aligned} \tag{1}$$

Note that $||\mathbf{u}'_i|| = ||\mathbf{u}'_{i+1}|| = 1$ and ${\mathbf{u}'_i}^\top \mathbf{u}'_{i+1} = 0$. Therefore, replacing $\mathbf{u}_i$ and $\mathbf{u}_{i+1}$ with $\mathbf{u}'_i$ and $\mathbf{u}'_{i+1}$ still satisfies EVD. We denote $\mathbf{V}' = [\mathbf{u}_1, \cdots, \mathbf{u}'_i, \mathbf{u}'_{i+1}, \cdots, \mathbf{u}_k]$. Then the perturbation can be represented as $\Delta \mathbf{L} = \mathbf{V}' \mathbf{\Lambda} {\mathbf{V}'}^\top - \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top = \lambda_i(\mathbf{u}'_i {\mathbf{u}'_i}^\top - \mathbf{u}_i \mathbf{u}_i^\top) + \lambda_{i+1}(\mathbf{u}'_{i+1}{\mathbf{u}'_{i+1}}^\top - \mathbf{u}_{i+1}\mathbf{u}_{i+1}^\top)$.

For sufficient small $\epsilon > 0$, we have:

$$\begin{aligned} & \min_{\mathbf{Q} \in \mathbf{O}(k)} ||(\mathbf{V} + \Delta \mathbf{V}) - \mathbf{V}\mathbf{Q}||_F \\ =& ||\mathbf{u}'_i - \mathbf{u}_i, \mathbf{u}'_{i+1} - \mathbf{u}_{i+1}||_F \\ =& ||(\sqrt{1-\epsilon^2} - 1)\mathbf{u}_i + \epsilon \mathbf{u}_{i+1}||_F + ||(\sqrt{1-\epsilon^2}-1)\mathbf{u}_{i+1} - \epsilon \mathbf{u}_i||_F \\ =& 4(1 - \sqrt{1-\epsilon^2}) \\ =& 2\epsilon^2 + o(\epsilon^2). \end{aligned} \tag{2}$$

For Fourier features with period $\frac{T}{2}$, we have:

$$\begin{aligned} & \forall_{\mathbf{Q} \in \mathbf{O}(k)} ||(\mathbf{V} + \Delta \mathbf{V})\rho(\boldsymbol{\lambda}_k) - \mathbf{V}\rho(\boldsymbol{\lambda}_k)||_F \\ =& || \left[\mathbf{u}'_i - \mathbf{u}_i, \mathbf{u}'_{i+1} - \mathbf{u}_{i+1}\right] \left[\rho(\lambda_i), \rho(\lambda_{i+1})\right]^\top ||_F \\ =& \sum_{t=1}^{T/2} || \sin(\lambda_i)(\mathbf{u}'_i - \mathbf{u}_i) + \sin(\lambda_{i+1})(\mathbf{u}'_{i+1} - \mathbf{u}_{i+1})||_F \\ &+ \sum_{t=1}^{T/2} || \cos(\lambda_i)(\mathbf{u}'_i - \mathbf{u}_i) + \cos(\lambda_{i+1})(\mathbf{u}'_{i+1} - \mathbf{u}_{i+1})||_F \\ \leq& T\left(||\mathbf{u}'_i - \mathbf{u}_i||_F + ||\mathbf{u}'_{i+1} - \mathbf{u}_{i+1}||_F\right) \\ \leq& T(2\epsilon^2 + o(\epsilon^2)), \end{aligned} \tag{3}$$

Next, we characterize $||\Delta \mathbf{L}||_F$:

$$\begin{aligned} & ||\Delta \mathbf{L}||_F \\ =& \left\| \lambda_i \left(\mathbf{u}'_i {\mathbf{u}'_i}^\top - \mathbf{u}_i \mathbf{u}_i^\top \right) + \lambda_{i+1} \left(\mathbf{u}'_{i+1} {\mathbf{u}'_{i+1}}^\top - \mathbf{u}_{i+1} \mathbf{u}_{i+1}^\top \right) \right\|_F \\ =& \left\| (\lambda_{i+1} - \lambda_i) \left[-\epsilon^2 \left(\mathbf{u}_i \mathbf{u}_i^\top - \mathbf{u}_{i+1}\mathbf{u}_{i+1}^\top \right) + \epsilon \sqrt{1-\epsilon^2} \left(\mathbf{u}_i \mathbf{u}_{i+1}^\top + \mathbf{u}_{i+1} \mathbf{u}_i^\top \right) \right] \right\|_{\mathrm{F}}^2 \\ =& (\lambda_{k+1} - \lambda_k)^2 \left(\epsilon^2 \left\| \mathbf{u}_k \mathbf{u}_{k+1}^\top + \mathbf{u}_{k+1} \mathbf{u}_k^\top \right\|_{\mathrm{F}}^2 + o\left(\epsilon^2\right) \right) \\ =& 2\left(\lambda_{k+1} - \lambda_k\right)^2 \left(\epsilon^2 + o\left(\epsilon^2\right) \right) \end{aligned} \tag{4}$$

Combining Equations (2) and (4), we have the lower bound of the changes of non-equivariant spectral features under small perturbations:

$$\min_{\mathbf{Q} \in \mathbf{O}(k)} ||(\mathbf{V} + \Delta \mathbf{V}) - \mathbf{V}\mathbf{Q}||_F \geq 0.99 \max_{1 \leq i \leq k} |\lambda_{i+1} - \lambda_i|^{-1} ||\Delta \mathbf{L}||_F + o(\epsilon), \tag{5}$$

which concludes the Lemma 1, *i.e.*, Lemma 3.4 in [37].

Combining Equations (3) and (4), we have the upper bound of the changes of equivariant spectral features under small perturbations:

$$\forall_{\mathbf{Q} \in \mathbf{O}(k)} ||(\mathbf{V} + \Delta \mathbf{V})\rho(\boldsymbol{\lambda}_k) - \mathbf{V}\rho(\boldsymbol{\lambda}_k)||_F \leq 0.99T \max_{1 \leq i \leq k} |\lambda_{i+1} - \lambda_i|^{-1} ||\Delta \mathbf{L}||_F + o(\epsilon), \tag{6}$$

which concludes the Theorem 2.

B Detailed Experimental Setup

In this section, we report the details of our experiments. Specifically, we first introduce some general settings in all experiments. Then we introduce the detailed setup of each experiment one by one.

B.1 General Settings

Optimizer. For all experiments, we use the Adam optimizer.

Environment. The environment in which we run experiments is:

- Linux version: 5.19.0-38-generic
- Operating system: Ubuntu 22.04.2
- CPU information: AMD EPYC 7313P 16-Core Processor
- GPU information: GeForce RTX 3090 (24 GB)

Resources. The addresses and licenses of all datasets are as follows:

- PubMed: `https://github.com/tkipf/pygcn` (MIT License)
- Wiki-CS: `https://github.com/pmernyei/wiki-cs-dataset` (MIT License)
- Facebook: `https://github.com/benedekrozemberczki/MUSAE` (GPL-3.0 license)
- arXiv: `https://github.com/snap-stanford/ogb` (MIT license)
- Flickr: `https://github.com/GraphSAINT/GraphSAINT` (MIT license)
- PPI: `https://github.com/mims-harvard/ohmnet` (MIT license)
- OGB-graph: `https://github.com/snap-stanford/ogb` (MIT license)
- ZINC-2M: `https://github.com/snap-stanford/pretrain-gnns` (MIT license)

Reproducibility. Our code is attached in the supplementary material.

B.2 Unsupervised Node Classification

Evaluation protocol. In the unsupervised node classification task, all methods are first trained with the corresponding self-supervised learning objectives. Then the learned representations are evaluated with a Logistic classifier with l_2 normalization. We evaluate the method every 10 epochs and the maximum epoch is set to 1000. For the mini-batch training, we set the batch size to 1024. The detailed statistics are shown in Table 1 and the hyperparameters are shown in Table 2.

Table 1: Statistics of unsupervised node classification datasets.

	Graphs	**Nodes**	**Edges**	**Features**	**Classes**
PubMed	1	19,717	88,648	500	3
Wiki-CS	1	11,701	216,123	300	10
Facebook	1	22,470	342,004	128	4
arXiv	1	169,343	1,116,243	128	40
Flickr	1	89,250	899,756	500	7
PPI	24	56,928	1,226,368	50	121

B.3 Unsupervised Graph Prediction

Evaluation protocol. In the unsupervised graph prediction task, we use the stand encoder, provided by OGB [2], as the spatial encoder of Sp^{2}GCL, which is a 5-layer GIN with hidden dimension $d = 300$. We use add pooling to learn graph-level representations and set the batch size to 32. For the spectral

---

[2] https://github.com/snap-stanford/ogb/blob/master/ogb/graphproppred/mol_encoder.py

Table 2: Statistics of unsupervised node classification datasets.

	# Eigenvectors (k)	Period (T)	lr	wd	Dropout
PubMed	30	20	1e-3	0	0
Wiki-CS	100	20	1e-3	0	0
Facebook	100	20	1e-3	0	0
arXiv	200	20	1e-3	0	0
Flickr	100	20	1e-3	0	0
PPI	50	20	1e-3	0	0

encoder, due to the relatively small sizes of the molecular graphs, we use all eigenvectors as the spectral features. We set the learning rate to 0.001 and the period to 10 for all datasets, and the number of training epochs is chosen among {20, 50, 80, 100, 150} using the validation set, as suggested by AD-GCL [27]. For the downstream evaluator, we use a Riger regressor for the regression tasks and a Logistic classifier for the binary classification tasks. The strength of l_2 normalization is grid searched among {0.001, 0.01, 0.1, 1, 10, 100, 1000} on the validation set for each dataset. The detailed statistics of the datasets are shown in Table 3.

Table 3: Statistics of unsupervised graph prediction datasets.

	Graphs	**Avg. Nodes**	**Avg. Edges**	**Classes**	**Task**	**Metric**
ogbg-molesol	1,128	13.3	13.7	1	Regression	RMSE
ogbg-mollipo	4,200	27.0	29.5	1	Regression	RMSE
ogbg-molfreesolv	642	8.7	8.4	1	Regression	RMSE
ogbg-molbace	1,513	34.1	36.9	1	Binary Class.	ROC-AUC
ogbg-molbbbp	2,039	24.1	26.0	1	Binary Class.	ROC-AUC
ogbg-molclintox	1,477	26.2	27.9	2	Binary Class.	ROC-AUC
ogbg-moltox21	7,831	18.6	19.3	12	Binary Class.	ROC-AUC
ogbg-molsider	1,427	33.6	35.4	27	Binary Class.	ROC-AUC

B.4 Transfer Learning

Evaluation protocol. For the transfer learning task, we use the same GIN encoder as [10]. In the pre-training stage, the learning rate is set to 0.001 and the number of training epochs is chosen from {20, 50, 80, 100} based on the validation set. Similarly, we use all eigenvalues and eigenvectors as the spectral features, and the period is set to 10. In the fine-tuning stage, we remove the self-supervised learning objective, and an additional linear projection layer is used on the output of the encoder for classification. The hyperparameters are the same as in the pre-training stage. The detailed statistics of the datasets are shown in Table 4.

Table 4: Statistics of transfer learning datasets.

	Graphs	**Utilization**	**Avg. Nodes**	**Avg. Edges**
ZINC-2M	Pre-Training	2,000,000	26.62	57.72
BBBP	Finetuning	2,039	24.06	51.90
Tox21	Finetuning	7,831	18.57	38.58
SIDER	Finetuning	1,427	33.64	70.71
ClinTox	Finetuning	1,477	26.15	55.76
BACE	Finetuning	1,513	34.08	73.71
HIV	Finetuning	41,127	25.51	54.93
MUV	Finetuning	93,087	24.23	52.55
ToxCast	Finetuning	8,576	18.78	38.52

B.5 Stability Experiment

We use the PyTorch-style pseudo code to explain how we generate the synthetic perturbations (Figure 2) and practical perturbations (Figure 3).

```
e, u = torch.linalg.eigh(L) # EVD

random_sign = 2*torch.randint(0, 2, (N,))- 1
sign_flip = torch.diag(random_sign).float()
coor_flip = torch.randperm(N)

u_sign = torch.mm(u, sign_flip)
u_basis = u.clone()[:, coor_flip]
```

Figure 2: Synthetic perturbations

```
e3, u3 = scipy.sparse.linalg.eigsh(
    L, k=100, which='SM', tol=1e-3)

e4, u4 = scipy.sparse.linalg.eigsh(
    L, k=100, which='SM', tol=1e-4)

e5, u5 = scipy.sparse.linalg.eigsh(
    L, k=100, which='SM', tol=1e-5)
```

Figure 3: Practical perturbations

C Matrix Form of EigenMLP

We give a detailed matrix form of Equation 6, from which we can see that the Fourier features of eigenvalues give different weights to the eigenvectors, thus making the model invariant to the rotation of coordinates and preserving good fitting ability.

$$\begin{aligned} &\underbrace{\begin{bmatrix} u_1^1 & u_2^1 & \cdots & u_k^1 \\ u_1^2 & u_2^2 & \cdots & u_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_1^N & u_2^N & \cdots & u_k^N \end{bmatrix}}_{\text{Eigenvectors, } N \times k} \times \underbrace{\begin{bmatrix} \cos(\lambda_1) & \sin(\lambda_1) & \cdots & \cos(T\lambda_1) & \sin(T\lambda_1) & \\ \cos(\lambda_2) & \sin(\lambda_2) & \cdots & \cos(T\lambda_2) & \sin(T\lambda_2) & \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \cos(\lambda_k) & \sin(\lambda_k) & \cdots & \cos(T\lambda_k) & \sin(T\lambda_k) & \end{bmatrix}}_{\text{Fourier features of eigenvalues, } k \times 2T} \\ &\times \underbrace{\begin{bmatrix} \alpha_1^1 & \alpha_2^1 & \cdots & \alpha_d^1 \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_d^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2T} & \alpha_2^{2T} & \cdots & \alpha_d^{2T} \end{bmatrix}}_{\text{Parameters of learnable matrix, } 2t \times d} = \underbrace{\begin{bmatrix} h_1^1 & h_2^1 & \cdots & h_d^1 \\ h_1^2 & h_2^2 & \cdots & h_d^2 \\ \vdots & \vdots & \ddots & \vdots \\ h_1^N & h_2^N & \cdots & h_d^N \end{bmatrix}}_{\text{Representations, } N \times d} \end{aligned} \tag{7}$$

D Pseudo Algorithm

In order to better demonstrate our algorithm, here we provide the pseudo algorithms of EigenMLP (Figure 4) and Sp^{2}GCL (Figure 5).

Figure 4: Pseudo Algorithm of EigenMLP

```
class EigenMLP(nn.Module)

    def __init__(self, k, d, T):
        # k: number of Eigenvectors
        # d: hidden dimension
        # T: period
        self.phi = nn.Sequential(nn.Linear(1, d), nn.ReLU(), nn.Linear(d, d))
        self.psi = nn.Sequential(nn.Linear(d, d), nn.ReLU(), nn.Linear(d, 1))
        self.mlp = nn.Sequential(nn.Linear(2*T, d), nn.ReLU(), nn.Linear(d, d))

    def forward(e, u):
        u = u.unsqueeze(-1)
        u = self.psi(self.phi(u) + self.phi(-u)).squeeze(-1)       # [N, k]

        T_term = torch.arange(0, T).float()
        T_e = e.unsqueeze(1) * T
        F_e = torch.cat([torch.sin(T_e), torch.cos(T_e)], dim=-1)  # [k, 2T]

        return self.mlp(torch.mm(u, F_e))
```

Figure 5: Pseudo Algorithm of Sp^{2}GCL

```
def Sp2GCL(g, x, e, u):
    # g: graph structures
    # x: node features
    # e: eigenvalues
    # u: eigenvectors
    x_a = GNN(g, x)
    x_e = EigenMLP(e, u)

    # For graph-level tasks
    # x_a = add_pool(g, x_a)
    # x_e = add_pool(g, x_e)

    h_a = spa_projection_head(x_a)
    h_e = spe_projection_head(x_e)

    h_a = F.normalize(h_a, dim=-1, p=2)
    h_e = F.normalize(h_e, dim=-1, p=2)

    logits = torch.mm(h_a, h_e.t())
    labels = torch.arange(h_a.size(0), dtype=torch.long)

    return 0.5 * F.cross_entropy(logits, labels) +
           0.5 * F.cross_entropy(logits.t(), labels)
```