

<b>Table of Contents</b>	<b>14</b>
<b>7 Appendix</b>	<b>15</b>
7.1 Notation Table . . . . .	15
7.2 More on Preliminaries . . . . .	16
7.2.1 Analysis of Real-world Graphs . . . . .	16
7.2.2 Ollivier-Ricci Curvature (ORC) . . . . .	17
7.2.3 Product Manifolds . . . . .	17
7.2.4 Kappa-Stereographic Model . . . . .	18
7.2.5 Spectral Graph Theory . . . . .	18
7.3 More on Cusp Laplacian . . . . .	18
7.3.1 Relevant Theorems for Cusp Laplacian . . . . .	19
7.4 Generalised Pageranks and GPRGNN . . . . .	21
7.4.1 Proof of Theorem 1 . . . . .	21
7.4.2 Why GPRGNN as the backbone of CUSP? . . . . .	22
7.5 Theorems for Curvature Encoding . . . . .	22
7.5.1 Proof of Definition 2 . . . . .	22
7.5.2 Proof of Theorem 2 . . . . .	23
7.6 Experimentation . . . . .	24
7.6.1 Datasets . . . . .	24
7.6.2 Baselines . . . . .	24
7.6.3 Experimental Results for Link Prediction . . . . .	25
7.6.4 Estimating Product Manifold Signature . . . . .	26
7.6.5 More Experimental Settings . . . . .	27

## 7 APPENDIX

## 7.1 NOTATION TABLE

Notation	Reference
$\mathbb{H}$	Hyperbolic manifold
$\mathbb{S}$	Spherical manifold
$\mathbb{E}$	Euclidean manifold
$\mathbb{P}^{d_{\mathcal{M}}}$	Product manifold of dimension $d_{\mathcal{M}}$
$\kappa$	Continuous manifold curvature
$\tilde{\kappa}$	Ollivier-Ricci Curvature (ORC)
$\tilde{\kappa}(x, y)$	ORC of edge $\{x, y\}$
$\tilde{\kappa}(x)$	ORC of node $x$
$m_x^\delta$	Probability mass assigned to node $x$ for ORC computation
$\delta_{ij}$	Kronecker delta function
$\mathcal{N}(x)$	Neighbourhood set of node $x$
$x \sim y$	This implies that $x$ and $y$ are adjacent nodes
$\delta$	ORC neighbourhood weighting parameter
$\mathbf{W}_1(\cdot)$	Wasserstein-1 distance
$d_{\mathcal{G}}(x, y)$	Shortest path (graph distance) between nodes $x$ and $y$ on graph $\mathcal{G}$
$\mathcal{M}^{\kappa_i, d_i}$	Constant-curvature manifold with dimension $d_i$ and curvature $\kappa_i$ . $\mathcal{M}_i \in \{\mathbb{H}, \mathbb{S}, \mathbb{E}\}$
$\tilde{\mathbf{L}}, \tilde{\mathbf{D}}, \tilde{\mathbf{A}}$	Cusp Laplacian operator; $\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}}$
$\tilde{\mathbf{L}}_n, \tilde{\mathbf{A}}_n$	Normalized Cusp Laplacian and Adjacency matrices; $\tilde{\mathbf{A}}_n = \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$
$\mathbf{L}, \mathbf{D}, \mathbf{A}$	Traditional graph Laplacian, Adjacency and Degree matrices; $\mathbf{L} = \mathbf{D} - \mathbf{A}$
$d_f$	Input graph feature dimension
$d_{\mathcal{M}}$	Total dimension of the product manifold
$d_{\mathcal{C}}$	Total dimension of the curvature embedding
$\mathbf{F} \in \mathbb{R}^{n \times d_f}$	Input feature matrix
$\exp_0^\kappa : \mathbb{R}^{d_{\mathcal{M}}} \rightarrow \mathcal{M}^\kappa$	Exponential map, to map from tangent plane (Euclidean) to the product manifold
$\oplus_\kappa$	<i>Mobius</i> addition
$\otimes_\kappa$	$\kappa$ -right-matrix-multiplication
$\boxtimes_\kappa$	$\kappa$ -left-matrix-multiplication
$\zeta(x)$	Final node representation for node $x$
$\epsilon_l$	Learnable weight of $l^{th}$ filter
$\zeta \in \mathbb{P}^{n \times (d_{\mathcal{M}} + d_{\mathcal{C}})}$	Matrix containing final node embeddings
$\beta_q$	Learnable weight of $q^{th}$ component manifold
$\tau_q$	Relative importance of manifold $\mathcal{M}_q$ in Cusp Pooling
$L$	Total number of filters
$\mathcal{Q}$	Total number of components in product manifold
$l$	Used to denote the $l^{th}$ filter
$\mathbf{Z}_{\mathbb{P}^{d_{\mathcal{M}}}}^{(L)(x)}$	The GPR-based node representation for filter with $L$ layers, on manifold $\mathbb{P}^{d_{\mathcal{M}}}$
$\kappa_{(q)}$	Manifold curvature of $q^{th}$ component in product manifold
$d_{(q)}$	Manifold dimension of $q^{th}$ component in product manifold
$\mathbf{H}_{\mathcal{M}_q^{\kappa_{(q)}, d_{(q)}}}^{(l)}$	Hidden state representation after $l$ layers of GPR, on $q^{th}$ manifold component with curvature $\kappa_{(q)}$ and dimension $d_{(q)}$
$\gamma_l$	GPR weight for $l^{th}$ layer in the filter, while propagation GPR score.
$\alpha$	Initialising parameter for GPR
Signature	$\mathbb{P}^{d_{\mathcal{M}}} = \times_{q=1}^{\mathcal{Q}} \mathcal{M}_q^{\kappa_{(q)}, d_{(q)}} = (\times_{h=1}^{\mathcal{H}} \mathbb{H}_h^{\kappa_{(h)}, d_{(h)}}) \times (\times_{s=1}^{\mathcal{S}} \mathbb{S}_s^{\kappa_{(s)}, d_{(s)}}) \times \mathbb{E}^{d_{(e)}}$
$\psi : V \rightarrow \mathbb{R}$	A function defined on vertex set $V$
$\Phi_{\mathbb{P}^{d_{\mathcal{C}}}}(\tilde{\kappa}(x))$	Curvature encoding on product manifold
$\mathcal{K}_{\mathbb{P}}(\tilde{\kappa}_a, \tilde{\kappa}_b)$	$\mathcal{K}_{\mathbb{P}}(\tilde{\kappa}_a, \tilde{\kappa}_b) := \langle \Phi_{\mathbb{P}^{d_{\mathcal{C}}}}(\tilde{\kappa}_a), \Phi_{\mathbb{P}^{d_{\mathcal{C}}}}(\tilde{\kappa}_b) \rangle$ is the curvature kernel
$\lambda_i$	$i^{th}$ eigenvalue of $\tilde{\mathbf{A}}_n$
$\tilde{\lambda}_i$	$i^{th}$ eigenvalue of $\tilde{\mathbf{L}}_n$
$f_\theta(\cdot) : \mathbb{R}^{d_f} \rightarrow \mathbb{R}^{d_{\mathcal{M}}}$	Neural network with parameter set $\{\theta\}$ that generates the hidden state features before feeding input to Cusp Filtering.
$g_\theta(\cdot) : \mathbb{R}^{d_{\mathcal{M}}} \rightarrow \mathbb{R}^{d_{\mathcal{C}}}$	Neural network projector used in curvature encoding

## 7.2 MORE ON PRELIMINARIES

### 7.2.1 ANALYSIS OF REAL-WORLD GRAPHS

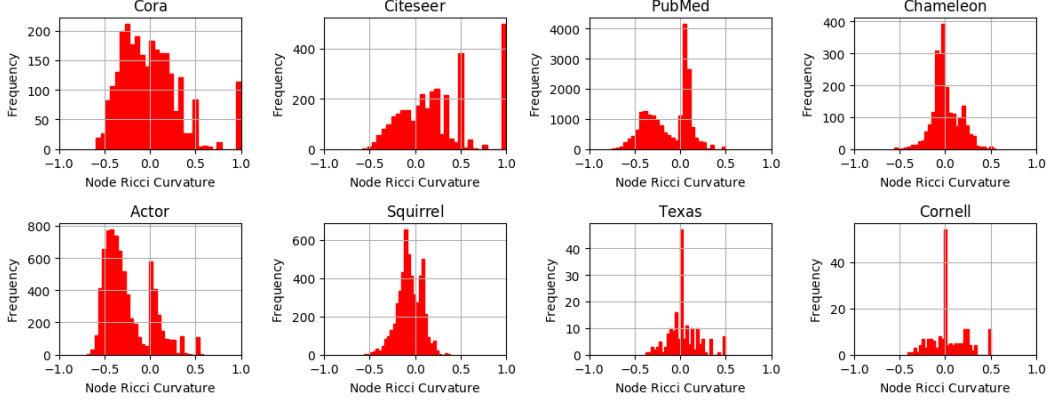


Figure 4: Distribution of Ollivier-Ricci curvatures  $\tilde{\kappa}(x, y)$  of **edges** across different datasets. The histograms illustrate the frequencies of edge-based Ollivier-Ricci curvature values for each dataset, *highlighting* the topological diversity in both homophilic and heterophilic settings, and hence the need of learning representations in manifolds with different curvatures.

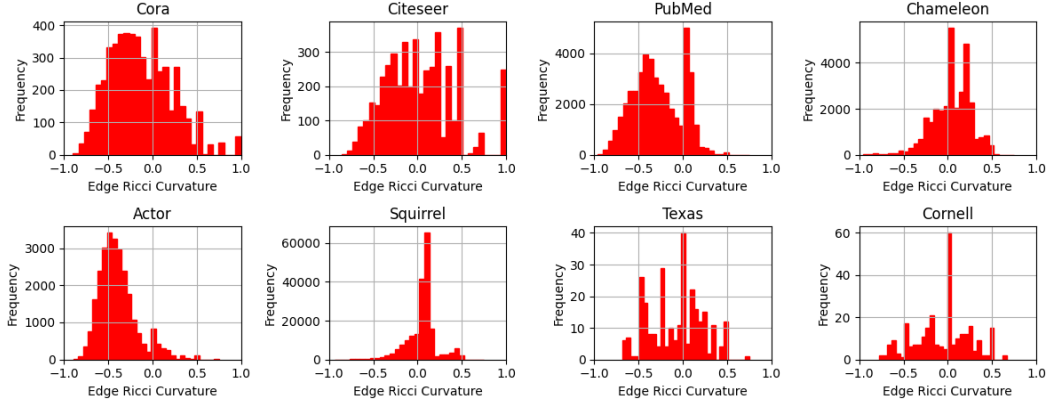


Figure 5: Distribution of Ollivier-Ricci curvatures  $\tilde{\kappa}(x)$  of **nodes** across different datasets.

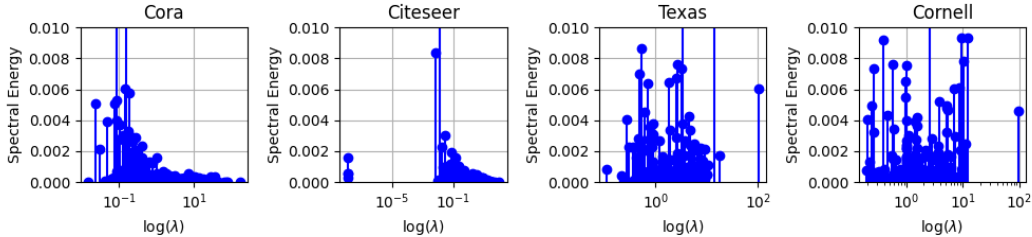


Figure 6: Distribution of **Spectral Energy** with the respective eigenvalues of the graph Laplacian. Spectral Energy,  $\mathcal{E}_i = f_i^2 / \sum_j f_j^2$ , where  $f_j$  is the  $j^{th}$  Fourier mode of the graph Laplacian. Low frequency implies *homophily* and high frequency components correspond to *heterophily*. These plots highlight the importance of capturing signals from different parts of the eigenspectrum for designing a GNN that works well across multiple tasks.

### 7.2.2 OLLIVIER-RICCI CURVATURE (ORC)

In an unweighted graph, the neighborhood of each node  $x$ , denoted as  $\mathcal{N}(x)$ , is assigned a probability distribution according to a lazy random walk formulation (Lin et al., 2011). Specifically, we define the distribution as follows:

$$m_z^\alpha(x) = \begin{cases} \alpha, & \text{if } z = x, \\ \frac{1-\alpha}{|\mathcal{N}(x)|}, & \text{if } z \in \mathcal{N}(x), \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Here,  $\alpha$  controls the probability that a random walk will remain at the current node, while the remaining probability mass  $(1 - \alpha)$  is uniformly distributed across the neighboring nodes. This formulation connects ORC with lazy random walks and influences the balance between local exploration and the likelihood of revisiting a node. In this work, we use  $\alpha = 0.5$ , meaning that equal probability mass is distributed between the node itself and its neighbors, striking a balance between *breadth-first* and *depth-first* search strategies. The choice of  $\alpha$  is crucial and depends on the topology of the graph. A smaller  $\alpha$  value encourages more local exploration, while a larger  $\alpha$  favors revisiting nodes, thereby promoting a “lazy” walk. For our experiments,  $\alpha = 0.5$  was chosen to reflect an equal probability mass distribution between the node and its neighbors.

■ **Computational Considerations.** Computing ORC can be computationally intensive due to the need to calculate the Wasserstein-1 distance ( $W_1$ ), between the neighborhood distributions of connected nodes. In a discrete setting, this corresponds to solving a linear program. Typically,  $W_1(m_x, m_y)$  between two nodes  $x$  and  $y$  is computed using the *Hungarian algorithm* (Kuhn, 1955), which has a cubic time complexity. However, this becomes prohibitively expensive as the graph size increases. Alternatively, the Wasserstein-1 distance can be approximated using the *Sinkhorn algorithm* (Sinkhorn & Knopp, 1967), which reduces the complexity to quadratic time. For this work, we employ the Sinkhorn approximation to compute ORC efficiently. Below, we provide an alternative to approximate ORC of an edge in linear time, in case of very large (million-scale) real-world graphs.

■ **Approximating ORC in Linear Time.** Even with the quadratic complexity of the Sinkhorn algorithm, scaling to large networks remains challenging. To address this, a linear-time combinatorial approximation of ORC can be employed, as suggested by Tian et al. (2023). This method approximates the Wasserstein distance by utilizing local structural information, making it much more computationally feasible. The approximation of ORC builds on classical bounds first introduced by Jost & Liu (2014). Let  $\#(x, y)$  denote the number of triangles formed by the edge  $(x, y)$ , and define  $a \wedge b = \min(a, b)$ ,  $a \vee b = \max(a, b)$  and  $d_x$  is the degree of node  $x$ . The following bounds on ORC can be derived for an edge  $e = x, y$ :

**Theorem 3** (Jost & Liu (2014)). *For an unweighted graph, the Ollivier-Ricci curvature of an edge  $e = x, y$  satisfies the following bounds:*

1. *Lower bound:*

$$\tilde{\kappa}(x, y) \geq - \left( 1 - \frac{1}{d_x} - \frac{1}{d_y} - \frac{\#(x, y)}{d_x \wedge d_y} \right)_+ - \left( 1 - \frac{1}{d_x} - \frac{1}{d_y} - \frac{\#(x, y)}{d_x \vee d_y} \right)_+ + \frac{\#(x, y)}{d_x \vee d_y}.$$

2. *Upper bound:*

$$\tilde{\kappa}(x, y) \leq \frac{\#(x, y)}{d_x \vee d_y}. \quad (10)$$

The ORC of an edge, can then be approximated as the arithmetic mean of these bounds:

$$\hat{\kappa}(x, y) := \frac{1}{2} \left( \kappa^{\text{upper}}(x, y) + \kappa^{\text{lower}}(x, y) \right). \quad (11)$$

The proof of these bounds has been detailed in Tian et al. (2023). This approximation is computationally efficient, with linear-time complexity, and can be parallelized easily across edges, making it suitable for large-scale graphs. The computation relies solely on local structural information, such as the degree of the nodes and the number of triangles.

### 7.2.3 PRODUCT MANIFOLDS

Let  $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k$  denote a set of smooth manifolds. Their Cartesian product forms a product manifold, denoted by  $\mathbb{P}$ , such that  $\mathbb{P} = \mathcal{M}_1 \times \mathcal{M}_2 \times \dots \times \mathcal{M}_k$ . Any point  $p \in \mathbb{P}$  is characterized by its coordinates  $p = (p_1, p_2, \dots, p_k)$ , where each  $p_i$  corresponds to a point on the individual manifold  $\mathcal{M}_i$ . Similarly, a tangent vector  $v \in \mathcal{T}_p \mathbb{P}$  can be expressed as  $v = (v_1, v_2, \dots, v_k)$ , where each  $v_i \in \mathcal{T}_{p_i} \mathcal{M}_i$  represents the projection of  $v$  in the tangent space of the respective component manifold  $\mathcal{M}_i$ . Optimization over manifolds requires the notion of taking steps along the manifold, which can be achieved by moving in the tangent space and mapping those movements back onto the manifold through the exponential map. The exponential map at a point  $p \in \mathbb{P}$ ,

denoted  $\exp_p : \mathcal{T}_p\mathbb{P} \rightarrow \mathbb{P}$ , allows for this transfer. For product manifolds, the exponential map decomposes into individual component exponential maps. Specifically, given a tangent vector  $v = (v_1, v_2, \dots, v_k)$  at  $p = (p_1, p_2, \dots, p_k) \in \mathbb{P}$ , the exponential map on  $\mathbb{P}$  can be expressed as:

$$\exp_p(v) = (\exp_{p_1}(v_1), \exp_{p_2}(v_2), \dots, \exp_{p_k}(v_k)) \quad (12)$$

#### 7.2.4 KAPPA-STEREOGRAPHIC MODEL

The  $\kappa$ -stereographic model (Bachmann et al., 2020) unifies Hyperbolic and Spherical geometries under gyrovector formalism. This model leverages the framework of gyrovector spaces to represent all three constant curvature geometries—hyperbolic, Euclidean, and spherical—simultaneously. Additionally, it facilitates smooth transitions between these constant curvature geometries, enabling the joint learning of space curvatures alongside the embeddings. It is a smooth manifold  $\mathcal{M}^{\kappa,d} = \{z \in \mathbb{R}^d \mid -\kappa\|z\|_2^2 < 1\}$ , whose origin is  $\mathbf{0} \in \mathbb{R}^d$ , equipped with a Riemannian metric  $g_z^\kappa = (\lambda_z^\kappa)^2 \mathbf{I}$ , where  $\lambda_z^\kappa$  is given by  $\lambda_z^\kappa = 2(1 + \kappa\|z\|_2^2)^{-1}$ . The Riemannian operations under this model are elaborated in the table below:

Operation	Formalism in $\mathbb{E}^d$	Unified formalism in $\kappa$ -stereographic model ( $\mathbb{H}^d/\mathbb{S}^d$ )
Distance Metric	$d_{\mathcal{M}}^\kappa(\mathbf{x}, \mathbf{y}) = \ \mathbf{x} - \mathbf{y}\ _2$	$d_{\mathcal{M}}^\kappa(\mathbf{x}, \mathbf{y}) = \frac{2}{\sqrt{ \kappa }} \tan_\kappa^{-1} \left( \sqrt{ \kappa } \ \mathbf{x} \oplus_\kappa \mathbf{y}\ _2 \right)$
Exp. Map	$\exp_{\mathbf{x}}^\kappa(\mathbf{v}) = \mathbf{x} + \mathbf{v}$	$\exp_{\mathbf{x}}^\kappa(\mathbf{v}) = \mathbf{x} \oplus_\kappa \left( \tan_\kappa \left( \sqrt{ \kappa } \frac{\lambda_{\mathbf{x}}^\kappa \ \mathbf{v}\ _2}{2} \right) \frac{\mathbf{v}}{\sqrt{ \kappa } \ \mathbf{v}\ _2} \right)$
Log. Map	$\log_{\mathbf{x}}^\kappa(\mathbf{y}) = \mathbf{x} - \mathbf{y}$	$\log_{\mathbf{x}}^\kappa(\mathbf{y}) = \frac{2}{\sqrt{ \kappa } \lambda_{\mathbf{x}}^\kappa} \tan_\kappa^{-1} \left( \sqrt{ \kappa } \ \mathbf{x} \oplus_\kappa \mathbf{y}\ _2 \right) \frac{-\mathbf{x} \oplus_\kappa \mathbf{y}}{\ \mathbf{x} \oplus_\kappa \mathbf{y}\ _2}$
Addition	$\mathbf{x} \oplus_\kappa \mathbf{y} = \mathbf{x} + \mathbf{y}$	$\mathbf{x} \oplus_\kappa \mathbf{y} = \frac{(1+2\kappa\mathbf{x}^T\mathbf{y}+\kappa\ \mathbf{y}\ _2^2)\mathbf{x}+(1-\kappa\ \mathbf{x}\ _2^2)\mathbf{y}}{1+2\kappa\mathbf{x}^T\mathbf{y}+\kappa^2\ \mathbf{x}\ _2^2\ \mathbf{y}\ _2^2}$

Table 7: Operations in Hyperbolic  $\mathbb{H}^d$ , Spherical  $\mathbb{S}^d$  and Euclidean space  $\mathbb{E}^d$ .

■  **$\kappa$ -right-matrix-multiplication.** Given a matrix  $\mathbf{X} \in \mathbb{R}^{n \times d}$  holding  $\kappa$ -stereographic embeddings in its rows and weights  $\mathbf{W} \in \mathbb{R}^{d \times e}$ , the Euclidean right multiplication can be written row-wise as  $(\mathbf{X}\mathbf{W})_{i\bullet} = \mathbf{X}_{i\bullet}\mathbf{W}$ . Then the  $\kappa$ -right-matrix-multiplication is defined row-wise as

$$(\mathbf{X} \otimes_\kappa \mathbf{W})_{i\bullet} = \exp_0^\kappa((\log_0^\kappa(\mathbf{X})\mathbf{W})_{i\bullet}) = \tan_\kappa(\alpha_i \tan_\kappa^{-1}(\|\mathbf{X}_{i\bullet}\|)) \frac{(\mathbf{X}\mathbf{W})_{i\bullet}}{\|(\mathbf{X}\mathbf{W})_{i\bullet}\|} \quad (13)$$

where  $\alpha_i = \frac{\|(\mathbf{X}\mathbf{W})_{i\bullet}\|}{\|\mathbf{X}_{i\bullet}\|}$  and  $\exp_0^\kappa$  and  $\log_0^\kappa$  denote the exponential and logarithmic map in the  $\kappa$ -stereo. model.

■  **$\kappa$ -left-matrix-multiplication.** Given a matrix  $\mathbf{X} \in \mathbb{R}^{n \times d}$  holding  $\kappa$ -stereographic embeddings in its rows and weights  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , the  $\kappa$ -left-matrix-multiplication is defined row-wise as

$$(\mathbf{A} \boxtimes_\kappa \mathbf{X})_{i\bullet} := \left( \sum_j A_{ij} \right) \otimes_\kappa m_\kappa(\mathbf{X}_{1\bullet}, \dots, \mathbf{X}_{n\bullet}; \mathbf{A}_{i\bullet}). \quad (14)$$

#### 7.2.5 SPECTRAL GRAPH THEORY

Graph Fourier Transform (GFT) (Sandryhaila & Moura, 2013) lays the foundation for Graph Neural Networks (GNNs). A GFT is defined using a *reference* operator  $\mathbf{R}$  which admits a spectral decomposition. Traditionally, in the case of GNNs, this reference operator has been the symmetric normalized Laplacian  $\mathbf{L}_n = \mathbf{I} - \mathbf{A}_n$  (Kipf & Welling, 2016). The graph Fourier transform of a signal  $\mathbf{f} \in \mathbb{R}^n$  is then defined as  $\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f} \in \mathbb{R}^n$ , and its inverse as  $\mathbf{f} = \mathbf{U}\hat{\mathbf{f}}$ . A graph filter is an operator that acts independently on the entire eigenspace of a diagonalisable and symmetric reference operator  $\mathbf{R}$ , by modulating their corresponding eigenvalues. A graph filter is defined via the graph filter function  $g(\cdot)$  operating on the reference operator as  $g(\mathbf{R}) = \mathbf{U}g(\mathbf{\Lambda})\mathbf{U}^\top$ .

#### 7.3 MORE ON CUSP LAPLACIAN

Spectral graph theory has shown significant progress in relating geometric characteristics of graphs to properties of spectrum of graph Laplacians and related matrices. Several variants of the graph Laplacian matrices have been shown to capture specific inductive biases for different tasks (Ko et al., 2023; Belkin et al., 2008; Jacobson & Sorkine-Hornung, 2012).

*Proof of Definition 1.* Say the function  $\psi : V \rightarrow \mathbb{R}$  is defined on the vertex set  $V$  of the graph. Suppose  $\psi$  describes a temperature distribution across a graph, where  $\psi(x)$  is the temperature at vertex  $x$ . According to Newton’s law of cooling (He, 2024), the heat transferred from node  $x$  to node  $y$  is proportional to  $\psi(x) - \psi(y)$

if nodes  $x$  and  $y$  are connected (if they are not connected, no heat is transferred). Consequently, the heat diffusion equation on the graph can be expressed as  $\frac{d\psi}{dt} = -\beta \sum_y \mathbf{A}_{xy}(\psi(x) - \psi(y))$ , where  $\beta$  is a constant of proportionality and  $\mathbf{A}$  denotes the adjacency matrix of the graph. Further insight can be gained by considering Fourier’s law of thermal conductance (Liu, 1990), which states that heat flow is inversely proportional to the resistance to heat transfer. ORC measures the transportation cost ( $\mathbf{W}_1(:, :)$ ) between the neighborhoods of two nodes, reflecting the effort required to transport mass between these neighborhoods (Bauer et al., 2011). We interpret this transportation cost as the *resistance* between nodes. The vital takeaway here is that – *Heat flow between two nodes in a graph is influenced by the underlying Ollivier-Ricci curvature (ORC) distribution*. The diffusion rate is faster on an edge with positive curvature (low resistance), and slower on an edge with negative curvature (high resistance). Thus, the ratio  $\mathcal{R}_{xy}^{res} = \frac{\mathbf{W}_1(m_x, m_y)}{d_G(x, y)}$  represents the *resistance* from node  $x$  to node  $y$ , i.e.  $\frac{d\psi_{xy}}{dt} \propto \frac{1}{\mathcal{R}_{xy}^{res}}$ . It can be observed that  $\frac{1}{\mathcal{R}_{xy}^{res}} = \frac{d_G(x, y)}{\mathbf{W}_1(m_x, m_y)} = \frac{1}{1 - \tilde{\kappa}(x, y)}$  (From the definition of ORC) would tend to infinity when  $\mathbf{W}_1(m_x, m_y) = 0$  (i.e.  $\tilde{\kappa}(x, y) = 1$ ). Thus, to ensure continuity, we create a new ratio as  $\frac{1}{\mathcal{R}_{xy}^\bullet} = e^{-\mathcal{R}_{xy}^{res}} = e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}}$ . Thus, we can modify the above heat flow equation as:

$$\begin{aligned}
\frac{d\psi}{dt} &= -\bar{\beta} \sum_y \frac{\mathbf{A}_{xy}(\psi(x) - \psi(y))}{\mathcal{R}_{xy}^\bullet} \quad (\text{Inversely proportional to } \mathcal{R}_{xy}^\bullet) \\
&= -\bar{\beta} \sum_y \mathbf{A}_{xy}(\psi(x) - \psi(y)) e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} = -\bar{\beta} \left( \psi(x) \sum_y \mathbf{A}_{xy} - \sum_y \mathbf{A}_{xy} \psi(y) \right) e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \\
&= -\bar{\beta} \left( \psi(x) \mathbf{D}_{xx} - \sum_y \psi(y) \mathbf{A}_{xy} \right) e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \quad (\because \mathbf{D}_{xx} = \sum_y \mathbf{A}_{xy}) \\
&= -\bar{\beta} \sum_y \left( \delta_{xy} e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \mathbf{D}_{xx} - e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \mathbf{A}_{xy} \right) \psi(y) \quad (\delta_{xy} \text{ is the Kronecker delta.}) \\
&= -\bar{\beta} \sum_y \left( e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \mathbf{L}_{xy} \right) \psi(y) \quad (\mathbf{L}, \mathbf{D}, \mathbf{A} \text{ are Laplacian, Degree and Adjacency matrices.}) \\
&= -\bar{\beta} \left( \mathbf{L} \odot e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \right) \psi = -\bar{\beta} \tilde{\mathbf{L}} \psi \quad (\odot \text{ is the element-wise product})
\end{aligned}$$

This gives us the standard heat equation on graphs. Here,  $\bar{\beta}$  is the updated constant of proportionality.  $\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{A}}$  is the **Cusp Laplacian** operator, where  $\tilde{\mathbf{D}}$  and  $\tilde{\mathbf{A}}$  are the updated Degree and Adjacency matrices, to represent that the graph is transformed under edge weights  $w_{xy} = \frac{1}{\mathcal{R}_{xy}^\bullet} = e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}}$ . Finally, our Cusp Laplacian operator can be written as ( $x \sim y$  means  $xy$  is an edge in the graph):

$$\tilde{\mathbf{L}}\psi(x) = \sum_{y \sim x} \bar{w}_{xy} (\psi(x) - \psi(y)) = \sum_{y \sim x} e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} (\psi(x) - \psi(y)) \quad (15)$$

■ **Why is  $e^{-\mathcal{R}_{xy}^{res}} = e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}}$  the right choice?** To mathematically justify that  $e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}}$  is an appropriate choice, we must verify its properties:

1. **Asymptotics.** As  $\tilde{\kappa}(x, y) \rightarrow 1$ ,  $e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \rightarrow 0$ , indicating that nodes with high positive curvature experience very fast heat diffusion (minimal resistance). Conversely, as  $\tilde{\kappa}(x, y) \rightarrow -1$ ,  $e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}} \rightarrow \frac{1}{\sqrt{e}}$ , meaning that nodes with high negative curvature have slow heat diffusion (higher resistance).
2. **Continuity.** The exponential function is smooth and continuous, ensuring that even small changes in the curvature result in smooth changes in the heat flow dynamics, which is crucial for stable numerical simulations and theoretical consistency.
3. **Monotonicity.** For  $\tilde{\kappa}(x, y) > 0$ ,  $e^{\frac{-1}{1 - \tilde{\kappa}(x, y)}}$  is a decreasing function with respect to  $\tilde{\kappa}(x, y)$ . This means as curvature increases, the resistance decreases, aligning with the physical intuition of heat flow.

□

### 7.3.1 RELEVANT THEOREMS FOR CUSP LAPLACIAN

**Theorem 4** (Positive Semidefiniteness of Cusp Laplacian). *The normalized Laplacian operator  $\tilde{\mathbf{L}}$  is positive semidefinite, i.e., for any real vector  $u \in \mathbb{R}^n$ , we have  $u^T \tilde{\mathbf{L}} u \geq 0$ .*

*Proof.* We start by showing that the normalized **Cusp Laplacian**

$$\tilde{\mathbf{L}}_n = \mathbf{I} - \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} = \mathbf{I} - \tilde{\mathbf{A}}_n \quad (16)$$

is positive semi-definite. Let  $\mathbf{u}$  be any real vector of unit norm and  $\mathbf{f} = \tilde{\mathbf{D}}^{-1/2} \mathbf{u}$ , then we have

$$\mathbf{u}^T \tilde{\mathbf{L}}_n \mathbf{u} = \mathbf{u}^T \mathbf{u} - \mathbf{u}^T \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \mathbf{u} = \sum_{x=1}^n \mathbf{u}_x^2 - \sum_{x,y=1}^n \mathbf{f}_x \mathbf{f}_y \tilde{\mathbf{A}}_{xy} \quad (17)$$

$$= \sum_{x=1}^n \tilde{\mathbf{D}}_{xx} \mathbf{f}_x^2 - \sum_{x,y=1}^n \mathbf{f}_x \mathbf{f}_y \tilde{\mathbf{A}}_{xy} = \frac{1}{2} \left( \sum_{x=1}^n \tilde{\mathbf{D}}_{xx} \mathbf{f}_x^2 - 2 \sum_{x,y=1}^n \mathbf{f}_x \mathbf{f}_y \tilde{\mathbf{A}}_{xy} + \sum_{y=1}^n \tilde{\mathbf{D}}_{yy} \mathbf{f}_y^2 \right) \quad (18)$$

$$= \frac{1}{2} \sum_{x,y=1}^n \tilde{\mathbf{A}}_{xy} (\mathbf{f}_x - \mathbf{f}_y)^2 = \frac{1}{2} \sum_{x,y=1}^n e^{\frac{-1}{1-\tilde{\kappa}(x,y)}} \mathbf{A}_{xy} (\mathbf{f}_x - \mathbf{f}_y)^2, \quad (19)$$

where the last step follows from the definition of the degree. We know that  $e^{\frac{-1}{1-\tilde{\kappa}(x,y)}} > 0 \forall \kappa(x,y)$ , hence our Cusp Laplacian is positive semidefinite.  $\square$

**Theorem 5.** The eigenvalues  $\{\tilde{\lambda}_i\}_{i=1}^n$  of the normalized Cusp Laplacian  $\tilde{\mathbf{L}}_n$  lie in the interval  $[0, 2]$ .

*Proof.* We begin by noting that Theorem 4 shows that the normalized **Cusp Laplacian**  $\tilde{\mathbf{L}}_n$  has real, non-negative eigenvalues, meaning we need only to prove that the largest eigenvalue, denoted as  $\lambda_n$ , is less than or equal to 2. Before moving to that, we show that 0 is indeed an eigenvalue of  $\tilde{\mathbf{L}}$  associated with the unit eigenvector  $\boldsymbol{\tau}$  where  $\boldsymbol{\tau} = \frac{\sqrt{\tilde{\mathbf{D}}_{ii}}}{\sqrt{\sum_v \tilde{\mathbf{D}}_{vv}}}$ .

Let  $\mathbf{1}$  be the all one vector. Then, a direct calculation reveals that

$$\tilde{\mathbf{L}}_{\text{sym}} \boldsymbol{\tau} = \boldsymbol{\tau} - \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \boldsymbol{\tau} = \boldsymbol{\tau} - \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{D}}^{1/2} \mathbf{1} \times \frac{1}{\sqrt{\sum_v \tilde{\mathbf{D}}_{vv}}} \quad (20)$$

$$= \boldsymbol{\tau} - \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{A}} \mathbf{1} \times \frac{1}{\sqrt{\sum_v \tilde{\mathbf{D}}_{vv}}} = \boldsymbol{\tau} - \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{D}} \mathbf{1} \times \frac{1}{\sqrt{\sum_v \tilde{\mathbf{D}}_{vv}}} \quad (21)$$

$$= \boldsymbol{\tau} - \tilde{\mathbf{D}}^{1/2} \mathbf{1} \times \frac{1}{\sqrt{\sum_v \tilde{\mathbf{D}}_{vv}}} = \boldsymbol{\tau} - \boldsymbol{\tau} = 0. \quad (22)$$

Combining this result with the positive semi-definite property of the Laplacian shows that 0 is indeed the smallest eigenvalue of  $\tilde{\mathbf{L}}_{\text{sym}}$  associated with the eigenvector  $\boldsymbol{\tau}$ . For the second part, using the Courant-Fischer theorem, we know that the largest eigenvalue can be expressed as:

$$\lambda_n = \max_{\mathbf{u} \neq 0} \frac{\mathbf{u}^T \tilde{\mathbf{L}}_n \mathbf{u}}{\mathbf{u}^T \mathbf{u}}.$$

Substituting the definition of the normalized Cusp Laplacian  $\tilde{\mathbf{L}}_n = \mathbf{I} - \tilde{\mathbf{A}}_n$  into this expression, and letting  $\mathbf{f} = \tilde{\mathbf{D}}^{-1/2} \mathbf{u}$ , we have:

$$\lambda_n = \max_{\mathbf{u} \neq 0} \frac{\mathbf{u}^T \tilde{\mathbf{D}}^{-1/2} \tilde{\mathbf{L}} \tilde{\mathbf{D}}^{-1/2} \mathbf{u}}{\mathbf{u}^T \mathbf{u}} = \max_{\mathbf{f} \neq 0} \frac{\mathbf{f}^T \tilde{\mathbf{L}} \mathbf{f}}{\mathbf{f}^T \tilde{\mathbf{D}} \mathbf{f}}.$$

The degree matrix, can be expressed in the quadratic form as  $\mathbf{f}^T \tilde{\mathbf{D}} \mathbf{f} = \sum_{x=1}^n \tilde{\mathbf{D}}_{xx} |\mathbf{f}_x|^2$ .

For the numerator involving  $\tilde{\mathbf{L}}$ , we expand the quadratic form:

$$\mathbf{f}^T \tilde{\mathbf{L}} \mathbf{f} = \frac{1}{2} \sum_{x,y=1}^n \tilde{\mathbf{A}}_{xy} (\mathbf{f}_x - \mathbf{f}_y)^2 = \frac{1}{2} \sum_{x,y=1}^n e^{\frac{-1}{1-\tilde{\kappa}(x,y)}} \mathbf{A}_{xy} (\mathbf{f}_x - \mathbf{f}_y)^2 \quad (23)$$

$$\leq \sum_{x,y=1}^n e^{\frac{-1}{1-\tilde{\kappa}(x,y)}} \mathbf{A}_{xy} (\mathbf{f}_x + \mathbf{f}_y)^2 \leq 2 \sum_{x=1}^n |\mathbf{f}_x|^2 \left( \sum_{y=1}^n \mathbf{A}_{xy} \right) = 2 \sum_{x=1}^n \mathbf{D}_{xx} |\mathbf{f}(x)|^2. \quad (24)$$

The last inequality follows from the fact that  $e^{\frac{-1}{1-\tilde{\kappa}(x,y)}} \rightarrow \frac{1}{\sqrt{e}}$  as  $\tilde{\kappa}(x,y) \rightarrow -1$  implies that it is always  $< 1$ . Thus, we can conclude that,  $\mathbf{f}^T \tilde{\mathbf{L}} \mathbf{f} \leq 2 \mathbf{f}^T \tilde{\mathbf{D}} \mathbf{f}$ , and the Rayleigh quotient is bounded  $\frac{\mathbf{f}^T \tilde{\mathbf{L}} \mathbf{f}}{\mathbf{f}^T \tilde{\mathbf{D}} \mathbf{f}} \leq 2$ . This shows that the largest eigenvalue of the normalized Cusp Laplacian  $\tilde{\mathbf{L}}_n$  is bounded by 2, completing the proof that the eigenvalues of  $\tilde{\mathbf{L}}_n$  are contained within the interval  $[0, 2]$ .  $\square$

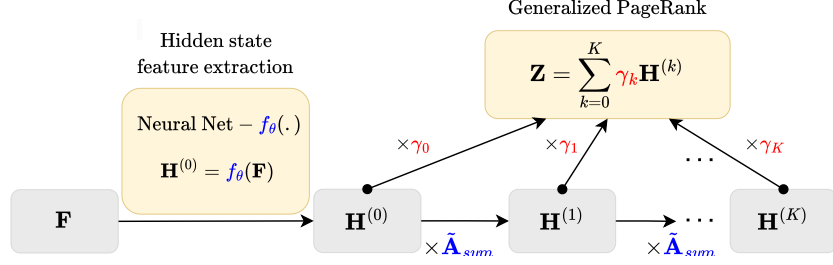


Figure 7: Architecture of GPRGNN.

#### 7.4 GENERALISED PAGERANKS AND GPRGNN

■ **Equivalence of the GPR method and polynomial graph filtering.** If we truncate the infinite series in the GPR definition at some integer  $K$ ,  $\sum_{k=0}^K \gamma_k \tilde{\mathbf{A}}_n^k$  becomes a polynomial graph filter of degree  $K$ . Consequently, optimizing the GPR weights is tantamount to optimizing the polynomial graph filter. It is important to note that any graph filter can be approximated using a polynomial graph filter, enabling the GPR method to handle a wide variety of node label patterns. Additionally, increasing  $K$  enhances the approximation of the optimal graph filter. This again illustrates the advantage of large-step propagation.

■ **GPRGNN architecture.** GPR-GNN initially derives hidden state features for each node and subsequently employs GPR to disseminate them. The GPR-GNN procedure can be represented as:

$$\hat{\mathbf{P}} = \text{softmax}(\mathbf{Z}), \mathbf{Z} = \sum_{k=0}^K \gamma_k \mathbf{H}^{(k)}, \mathbf{H}^{(k)} = \tilde{\mathbf{A}}_{sym} \mathbf{H}^{(k-1)}, \mathbf{H}_i^{(0)} = f_\theta(\mathbf{X}_i), \quad (25)$$

Here,  $f_\theta(\cdot)$  denotes a neural network parametrized by  $\{\theta\}$ , which produces the hidden state features  $\mathbf{H}^{(0)}$ . The GPR weights  $\gamma_k$  are optimized alongside  $\{\theta\}$  in an end-to-end manner. The GPR-GNN model is straightforward to interpret: As previously mentioned, GPR-GNN is capable of adaptively managing the contribution of each propagation step to fit the node label pattern. Analyzing the trained GPR weights also aids in understanding the topological properties of a graph, such as identifying the optimal polynomial graph filter.

##### 7.4.1 PROOF OF THEOREM 1

We first state the formal version of Theorem 1

**Theorem 6** (Formal version of Theorem 1). *Assume the graph  $G$  is connected. Let  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$  and  $\tilde{\lambda}_1 \leq \tilde{\lambda}_2 \leq \dots \leq \tilde{\lambda}_n$  be the eigenvalues of  $\tilde{\mathbf{A}}_n$  and  $\tilde{\mathbf{L}}_n$ , respectively. If  $\gamma_l \geq 0 \forall l \in \{0, 1, \dots, L\}$ ,  $\sum_{l=0}^L \gamma_l = 1$  and  $\exists l' > 0$  such that  $\gamma_{l'} > 0$ , then  $\left| \frac{g_{\gamma, L}(\lambda_i)}{g_{\gamma, L}(\lambda_1)} \right| < 1 \forall i \geq 2$ . Also, if  $\gamma_l = (-\alpha)^l$ ,  $\alpha \in (0, 1)$  and  $L \rightarrow \infty$ , then  $\left| \frac{\lim_{L \rightarrow \infty} g_{\gamma, L}(\lambda_i)}{\lim_{L \rightarrow \infty} g_{\gamma, L}(\lambda_1)} \right| > 1 \forall i \geq 2$ .*

1. Note that  $\left| \frac{g_{\gamma, L}(\lambda_i)}{g_{\gamma, L}(\lambda_1)} \right| < 1 \forall i \geq 2$  implies the **low-pass case** as after applying the graph filter  $g_{\gamma, L}$ , the lowest frequency component (correspond to  $\lambda_1$ ) further dominates.
2. **Unfiltered case.** Recall that in the unfiltered case, we do not multiply with  $\tilde{\mathbf{A}}_n$ . It can also be viewed as multiplying the identity matrix  $\mathbf{I}$ , where the eigenvalue ratio is  $\frac{|\lambda_i|}{|\lambda_1|} = 1$ . Hence  $g_{\gamma, L}$  acts like a low pass filter in this case.
3. In contrast,  $\left| \frac{\lim_{L \rightarrow \infty} g_{\gamma, L}(\lambda_i)}{\lim_{L \rightarrow \infty} g_{\gamma, L}(\lambda_1)} \right| > 1 \forall i \geq 2$  implies that after applying the graph filter, the lowest frequency component (correspond to  $\lambda_1$ ) no longer dominates. This corresponds to the **high pass filter** case.

**Proof.** We start with the **low pass filter result**. From Theorem 5, we know that  $0 \leq \tilde{\lambda}_1 \leq \tilde{\lambda}_2 \leq \dots \leq \tilde{\lambda}_n \leq 2$ . Given the spectrum of  $\tilde{\mathbf{A}}_n$ , we know that  $-\tilde{\mathbf{A}}_n$  has spectrum negatives of  $\tilde{\mathbf{A}}_n$ , and  $\mathbf{I} - \tilde{\mathbf{A}}_n$  adds one to each eigenvalue of  $-\tilde{\mathbf{A}}_n$ . Hence,  $1 \geq \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq -1$  follows directly. Now, we know that  $\lambda_1 = 1$  and  $|\lambda_i| < 1, \forall i \geq 2$ . Further, we have assumed that  $g_{\gamma, L}(\lambda_1) = \sum_{l=0}^L \gamma_l = 1$ . Hence, proving Theorem 6 is equivalent to show

$$|g_{\gamma, L}(\lambda_i)| < 1 \forall i \geq 2.$$



This is obvious since  $g_{\gamma,L}(\lambda) = \sum_{l=0}^L \gamma_l \lambda^l$  is a polynomial of order  $L$  with nonnegative coefficients. It is easy to check that  $\forall l \geq 1, |\lambda|^l < 1, \forall |\lambda| < 1$ . Combine with the fact that all  $\gamma_k$ 's are nonnegative we have

$$|g_{\gamma,L}(\lambda_i)| \leq \sum_{l=0}^L \gamma_l |\lambda|^l = \sum_{l=0}^L \gamma_l |\lambda|^l \stackrel{(a)}{\leq} \sum_{l=0}^L \gamma_l = 1.$$

Finally, note that the only possibility that the equality holds is  $\gamma_l = \delta_{0,l}$  since  $\forall l \geq 1, |\lambda|^l < 1, \forall |\lambda| < 1$ . However, by assumption  $\sum_{l=0}^L \gamma_l = 1$  and  $\exists l' > 0$  such that  $\gamma_{l'} > 0$  we know that this is impossible. Hence (a) is a strict inequality  $<$ .

For the **high pass filter result**, it can be observed that:

$$\lim_{L \rightarrow \infty} g_{\gamma,L}(\lambda) = \lim_{L \rightarrow \infty} \sum_{l=0}^L \gamma_l \lambda^l = \lim_{L \rightarrow \infty} \sum_{l=0}^L (-\alpha \lambda)^l = \frac{1}{1 + \alpha \lambda},$$

where the last step is due to the fact that  $\alpha \in (0, 1)$  and thus  $\lim_{L \rightarrow \infty} (-\alpha \lambda)^L = 0, \forall |\lambda| \leq 1$ . Thus we have

$$\left| \frac{\lim_{L \rightarrow \infty} g_{\gamma,L}(\lambda_i)}{\lim_{L \rightarrow \infty} g_{\gamma,L}(\lambda_1)} \right| = \left| \frac{1 + \alpha}{1 + \alpha \lambda_i} \right| \stackrel{(b)}{>} 1 \quad \forall i \geq 2.$$

The strict inequalities (b) is from the fact that  $|\lambda_i| < 1, \forall i \geq 2$ . Notably,  $\sup_{\lambda \in [-1, -1)} \frac{1}{1 + \alpha \lambda}$  happens at the boundary  $\lambda = -1$ , which corresponds to the bipartite graph. It further shows that the graph filter with respect to the choice  $\gamma_l = (-\alpha)^l$  emphasizes high frequency components and thus it is indeed acting as a high pass filter.  $\square$

#### 7.4.2 WHY GPRGNN AS THE BACKBONE OF CUSP?

In this section, we elaborate on why GPR is an ideal backbone when compared to other Spectral GNNs.

1. **Adaptive Filter Design.** GPR learns the filter coefficients directly, allowing the spectral response to adapt to the task and dataset. This flexibility is critical for modeling both homophilic and heterophilic graphs.
2. **Universality.** Unlike fixed low-pass filters like ChebNet, which excel primarily in homophilic settings, GPR's learnable filters enable it to balance low-pass and high-pass components, making it suitable for both homophilic and heterophilic graphs. This is one of the main goals of our paper - to achieve superior performance on homophilic and heterophilic tasks. Fixed polynomial filters in ChebNet and Bernstein-based methods approximate spectral responses up to a fixed order, limiting their ability to model complex spectral properties.
3. **GPRGNN escapes oversmoothing.** GPR weights are adaptively learnable, which allows GPR-GNN to avoid over-smoothing and trade node and topology feature informativeness. See Section 4 of Chien et al. (2020) for more theoretical analysis on the same and proofs, which is beyond the scope of this work. GPR not only mitigates feature over-smoothing, but also works on highly diverse node label patterns (See Section 4 and 5 of Chien et al. (2020)).
4. **Capturing node features and graph topology.** In many important graph data processing applications, the acquired information includes both node features and observations of the graph topology. GPRGNN jointly optimizes node feature and topological information extraction, regardless of the extent to which the node labels are homophilic or heterophilic.
5. **Filter Bank Construction.** Using GPR based spectral filters, helps us to effectively construct a filter bank where each adaptive filter contributes to a specific spectral profile, enabling the model to aggregate information across different spectral bands. This approach captures diverse patterns in node features and topology, unlike ChebNet or Bernstein-based methods, which rely on fixed polynomial approximations and lack such flexibility.

### 7.5 THEOREMS FOR CURVATURE ENCODING

**Theorem 7** (Bochner's Theorem). (Moeller et al., 2016) A continuous, translation-invariant kernel  $\mathcal{K}(\mathbf{x}, \mathbf{y}) = \Psi(\mathbf{x} - \mathbf{y})$  on  $\mathbb{R}^d$  is positive definite if and only if there exists a non-negative measure on  $\mathbb{R}$  such that  $\Psi$  is the Fourier transform of the measure.

#### 7.5.1 PROOF OF DEFINITION 2

**Proof.** Using the Bochner's theorem stated above, our curvature kernel  $\mathcal{K}_{\mathbb{R}}$  has the expression:

$$\mathcal{K}_{\mathbb{R}}(\tilde{\kappa}_a, \tilde{\kappa}_b) = \Psi_{\mathbb{R}}(\tilde{\kappa}_a, \tilde{\kappa}_a) = \int_{\mathbb{R}} e^{i\omega(\tilde{\kappa}_a - \tilde{\kappa}_b)} p(\omega) d\omega = \mathbb{E}_{\omega}[\xi_{\omega}(\tilde{\kappa}_a) \xi_{\omega}(\tilde{\kappa}_b)^*], \quad (26)$$

where  $\xi_\omega(\tilde{\kappa}) = e^{i\omega\tilde{\kappa}}$ . Since kernel  $\mathcal{K}_\mathbb{R}$  and measure  $p(\omega)$  are real, we extract the real part of (26):

$$\mathcal{K}_\mathbb{R}(\tilde{\kappa}_a, \tilde{\kappa}_b) = \mathbb{E}_\omega [\cos(\omega(\tilde{\kappa}_a - \tilde{\kappa}_b))] = \mathbb{E}_\omega [\cos(\omega\tilde{\kappa}_a) \cos(\omega\tilde{\kappa}_b) + \sin(\omega\tilde{\kappa}_a) \sin(\omega\tilde{\kappa}_b)]. \quad (27)$$

The above formulation suggests approximating the expectation by the Monte Carlo integral, i.e.  $\mathcal{K}_\mathbb{R}(\tilde{\kappa}_a, \tilde{\kappa}_b) \approx \frac{1}{d_C} \sum_{i=1}^{d_C} \cos(\omega_i\tilde{\kappa}_a) \cos(\omega_i\tilde{\kappa}_b) + \sin(\omega_i\tilde{\kappa}_a) \sin(\omega_i\tilde{\kappa}_b)$ , with  $\omega_1, \dots, \omega_{d_C} \stackrel{\text{i.i.d.}}{\sim} p(\omega)$ . Therefore, we propose the finite dimensional functional mapping to  $\mathbb{R}^{d_C}$  as:

$$\Phi_{\mathbb{R}^{d_C}}(\tilde{\kappa}) = \sqrt{\frac{1}{d_C}} [\cos(\omega_1\tilde{\kappa}), \sin(\omega_1\tilde{\kappa}), \dots, \cos(\omega_{d_C}\tilde{\kappa}), \sin(\omega_{d_C}\tilde{\kappa})] \quad (28)$$

The unknown probability distribution  $p(\omega)$  is estimated using the inverse cumulative distribution function (CDF) transformation as in Xu et al. (2020). Since our GNN is operating in the mixed-curvature space, we must map our defined curvature kernel based representations to the product manifold. We do so using the exponential map, for a node  $x$  with ORC curvature  $\tilde{\kappa}(x)$  as:

$$\Phi_{\mathbb{P}^{d_C}}(\tilde{\kappa}(x)) = g_\theta \left( \left\|_{q=1}^Q \exp_{\mathbf{0}}^{\kappa(q)}(\Phi_{\mathbb{R}^{d_C}}(\tilde{\kappa}(x))) \right\| \right) \quad (29)$$

$$= g_\theta \left( \left\|_{q=1}^Q \Phi_{\mathcal{M}_q^{\kappa(q), d(q)}}(\tilde{\kappa}(x)) \right\| \right) \quad (30)$$

where  $\exp_{\mathbf{0}}^{\kappa(q)} : \mathbb{R}^{d_C} \rightarrow \mathcal{M}_q^{\kappa(q), d(q)}$  denotes the exponential map on the  $q^{th}$  component manifold with curvature  $\kappa(q)$ ,  $\|$  is the concatenation operator and  $g_\theta : \mathbb{P}^{d_f} \rightarrow \mathbb{P}^{d_C}$  is a Riemannian projector. We need  $g_\theta$  because we maintain a single product manifold for CUSP with total dimension  $d_f$ . So, upon taking the exponential map with respect to this product manifold, we are required to project the curvature embeddings to the required dimension  $d_C$ .  $\square$

### 7.5.2 PROOF OF THEOREM 2

**Proof.** We begin by recalling that in Euclidean space, the curvature kernel  $\mathcal{K}_\mathbb{R}$  is:

$$\mathcal{K}_\mathbb{R}(\tilde{\kappa}_a, \tilde{\kappa}_b) = \langle \Phi_{\mathbb{R}^{d_C}}(\tilde{\kappa}_a), \Phi_{\mathbb{R}^{d_C}}(\tilde{\kappa}_b) \rangle = \Psi_\mathbb{R}(\tilde{\kappa}_a - \tilde{\kappa}_b).$$

The key property here is translation invariance:

$$\mathcal{K}_\mathbb{R}(\tilde{\kappa}_a + \tilde{c}, \tilde{\kappa}_b + \tilde{c}) = \mathcal{K}_\mathbb{R}(\tilde{\kappa}_a, \tilde{\kappa}_b) = \Psi_\mathbb{R}(\tilde{\kappa}_a - \tilde{\kappa}_b).$$

Next, we move to the product manifold  $\mathbb{P}^{d_C}$ , which consists of multiple components of different curvatures, such as hyperbolic, spherical, and Euclidean spaces.

For each component manifold  $\mathcal{M}_q^{\kappa(q), d(q)}$  with curvature  $\kappa(q)$ , the stereographic inner product  $\langle \cdot, \cdot \rangle_{\mathbf{x}}^\kappa : \mathcal{T}_x \mathcal{M}_\kappa^n \times \mathcal{T}_x \mathcal{M}_\kappa^n \rightarrow \mathbb{R}$ , is defined on the tangent plane of the Riemannian manifold as:

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{x}}^\kappa = \mathbf{u}^T \mathbf{g}_\mathbf{x}^\kappa \mathbf{v} = (\lambda_\mathbf{x}^\kappa)^2 \langle \mathbf{u}, \mathbf{v} \rangle,$$

where the conformal factor  $\lambda_\mathbf{x}^\kappa$  is defined as:

$$\lambda_\mathbf{x}^\kappa = \frac{2}{1 + \kappa \|\mathbf{x}\|_2^2}.$$

This conformal factor modulates the stereographic projection in the curved space, and it ensures that distances are mapped correctly in the manifold space. Now, consider the inner product between the curvature embeddings  $\Phi_{\mathbb{P}^{d_C}}(\tilde{\kappa}_a)$  and  $\Phi_{\mathbb{P}^{d_C}}(\tilde{\kappa}_b)$  in the mixed-curvature space.

$$\mathcal{K}_\mathbb{P}(\tilde{\kappa}_a, \tilde{\kappa}_b) = \sum_{q=1}^Q \langle \Phi_{\mathcal{M}_q^{\kappa(q), d(q)}}(\tilde{\kappa}_a), \Phi_{\mathcal{M}_q^{\kappa(q), d(q)}}(\tilde{\kappa}_b) \rangle_{\kappa(q)},$$

where each component manifold  $\mathcal{M}_q^{\kappa(q), d(q)}$  contributes to the overall inner product in the product manifold  $\mathbb{P}^{d_C}$ . Using the stereographic inner product in each component, we can write:

$$\mathcal{K}_\mathbb{P}(\tilde{\kappa}_a, \tilde{\kappa}_b) = \sum_{q=1}^Q \left( \lambda_\mathbf{x}^{\kappa(q)} \right)^2 \langle \Phi_{\mathbb{R}^{d_C}}(\tilde{\kappa}_a), \Phi_{\mathbb{R}^{d_C}}(\tilde{\kappa}_b) \rangle.$$

We now need to show that translation invariance holds in the mixed-curvature product manifold. Since the conformal factor  $\lambda_\mathbf{x}^\kappa$  depends only on the norm  $\|\mathbf{x}\|_2$ , any translation by a constant  $\tilde{c}$  does not affect the relative difference between curvature embeddings. Specifically, for any constant shift  $\tilde{c}$ :

$$\mathcal{K}_\mathbb{P}(\tilde{\kappa}_a + \tilde{c}, \tilde{\kappa}_b + \tilde{c}) = \sum_{q=1}^Q \left( \lambda_\mathbf{x}^{\kappa(q)} \right)^2 \Psi_\mathbb{R}((\tilde{\kappa}_a + \tilde{c}) - (\tilde{\kappa}_b + \tilde{c})) = \Psi_\mathbb{P}(\tilde{\kappa}_a - \tilde{\kappa}_b).$$

Thus, the kernel in the mixed-curvature space remains invariant to translation, completing the proof.  $\square$

## 7.6 EXPERIMENTATION

### 7.6.1 DATASETS

The performance of CUSP is evaluated over eight benchmark datasets for two primary tasks: Node Classification (NC) and Link Prediction (LP). These datasets encompass both homophilic and heterophilic domains. Detailed descriptions of each dataset are provided below.

1. *Citation Networks*. **Cora**, **PubMed**, and **Citeseer** (Sen et al., 2008; Yang et al., 2016) are citation networks in which nodes symbolize research papers, and edges denote citation links between them. Each node is labeled with its subject category. This dataset is commonly utilized for node classification because of its pronounced *homophilic* properties.
2. *Wikipedia graphs*. **Chameleon** and **Squirrel** (Rozemberczki et al., 2021) are *heterophilic* graphs derived from Wikipedia articles. Nodes represent articles, and edges represent hyperlinks between them. Node labels correspond to website traffic levels.
3. *Actor Co-occurrence Network*. **Actor** (Tang et al., 2009) is a heterophilic graph dataset where nodes depict actors and edges signify co-occurrences on the same Wikipedia page. The node labels correspond to the professional background of the actors.
4. *Webpage graphs*. **Texas** and **Cornell**<sup>4</sup> are parts of the WebKB dataset, and are sparsely connected heterophilic graphs. Here, nodes represent web pages, and edges represent hyperlinks between them. Labels reflect different types of webpages.

### 7.6.2 BASELINES

This part offers an in-depth discussion of the baseline models against which CUSP is compared. We classify the baseline models into three categories: **Spatial**, **Riemannian**, and **Spectral** methods, which each address a distinct facet of graph neural network architectures.

■ **Spatial baselines**. The first kind of baselines includes the traditional spatial methods, which directly operate on the node features and their immediate neighborhoods.

1. **GCN** (Kipf & Welling, 2016). Graph Convolutional Networks (GCNs) represent one of the foundational graph neural networks that utilize spectral graph convolution in the spatial domain. They derive node embeddings by combining features from neighboring nodes via a linear combination involving the adjacency matrix and the nodes' features.
2. **GAT** (Veličković et al., 2017). Graph Attention Network (GAT) introduces attention mechanisms to graph neural networks. Each node assigns learnable attention weights to its neighbors and aggregates their features based on these weights.
3. **GraphSAGE** (Hamilton et al., 2017). GraphSAGE is an inductive technique designed to learn node embeddings by sampling and aggregating features from a *fixed* set of neighboring nodes, instead of processing the entire graph. This method enables GraphSAGE to create embeddings for nodes not encountered during training by using efficient neighborhood sampling.

■ **Riemannian Baselines**. Riemannian models function within non-Euclidean spaces (such as hyperbolic or spherical manifolds) and are tailored for graph data characterized by intricate geometric properties (e.g. hierarchical or cyclic structures).

1. **HGCN** (Chami et al., 2019). Hyperbolic Graph Convolutional Networks utilize *hyperbolic geometry* to represent the hierarchical and tree-like characteristics of graphs. This approach extends GCN to hyperbolic space by introducing a hyperbolic variant of the convolutional operation. It is especially suitable for datasets that exhibit hierarchical or tree-like configurations.
2. **HGAT** (Zhang et al., 2021b). Hyperbolic Graph Attention Network (HGAT) extends Graph Attention Networks (GAT) into hyperbolic space by integrating attention mechanisms with hyperbolic geometry, and calculates the attention weights among the nodes in the hyperbolic space to enhance the aggregation of features.
3.  **$\kappa$ GCN** (Bachmann et al., 2020).  $\kappa$ GCN allows for learning the curvature of each node in a graph and generalizes GCN to operate in mixed-curvature spaces. The curvature parameter  $\kappa$  determines whether a node lies in hyperbolic, spherical, or Euclidean space. By learning curvature adaptively,  $\kappa$ GCN offers flexibility in modeling graphs with regions of different geometries, providing a better fit for graphs with complex structures.

<sup>4</sup><http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb>

4. **QGCN** (Xiong et al., 2022). Pseudo-Riemannian GCN extends a GCN to a pseudo-Riemannian manifold, enabling functionality in mixed-curvature spaces. This network is capable of modeling graph regions with both positive and negative curvature.
5. **SelfMGNN** (Sun et al., 2022). SelfMGNN generates embeddings within a mixed-curvature space through self-supervision. It dynamically allocates varied curvatures to different regions of the graph by utilizing a mixed-curvature embedding space. This approach incorporates both self-supervision and mixed-curvature learning to improve performance on heterogeneous graphs.

■ **Spectral Baselines.** These techniques utilize the eigenvalues of either the graph Laplacian or adjacency matrix to establish convolutional filters that function in the frequency domain.

1. **ChebyNet** (Defferrard et al., 2016). ChebyNet implements spectral convolutions through a polynomial approximation of the graph Laplacian, sidestepping the expensive process of eigenvalue decomposition. Instead, it approximates the convolution using Chebyshev polynomials. This approach allows ChebyNet to execute localized graph convolutions efficiently, making it well-suited for handling larger graphs.
2. **BernNet** (He et al., 2021). BernNet employs Bernstein polynomials to approximate graph filters, providing flexible management over the filter’s frequency response. This method extends polynomial-based graph filters and is adaptable to various frequency elements in graphs.
3. **GPRGNN** (Chien et al., 2020). The Generalized PageRank Graph Neural Network (GPRGNN) builds upon the Personalized PageRank (PPR) approach, integrating it into the framework of graph neural networks. It propagates node features through the graph by using a weighted sum of adjacency matrix powers, dynamically adjusting to both homophilic and heterophilic graphs.
4. **FiGURe** (Ekbote et al., 2024). FiGURe employs adaptive filters to capture various sections of the graph spectrum, enabling it to learn both high-pass and low-pass filters specific to the task. It dynamically selects the optimal filter bank to accurately represent the graph’s architecture.

### 7.6.3 EXPERIMENTAL RESULTS FOR LINK PREDICTION

Baseline	Cora	Citeseer	PubMed	Chameleon	Actor	Squirrel	Texas	Cornell	Av. $\Delta$ Gain
GCN	88.54 $\pm$ 0.51	85.42 $\pm$ 0.89	91.31 $\pm$ 0.73	86.07 $\pm$ 0.64	85.12 $\pm$ 0.78	90.01 $\pm$ 0.15	69.08 $\pm$ 0.99	73.09 $\pm$ 0.92	9.58
GAT	85.45 $\pm$ 0.66	87.23 $\pm$ 0.11	87.65 $\pm$ 0.04	88.99 $\pm$ 0.13	87.33 $\pm$ 0.08	90.23 $\pm$ 0.14	68.79 $\pm$ 0.72	75.12 $\pm$ 0.77	9.31
SAGE	87.12 $\pm$ 0.82	90.71 $\pm$ 0.65	90.09 $\pm$ 0.90	90.01 $\pm$ 0.58	86.06 $\pm$ 0.73	91.02 $\pm$ 0.61	76.54 $\pm$ 0.69	77.98 $\pm$ 0.88	6.97
HGCN	91.63 $\pm$ 0.55	<b>94.13<math>\pm</math>0.67</b>	91.04 $\pm$ 0.79	91.45 $\pm$ 0.62	90.01 $\pm$ 0.80	92.34 $\pm$ 0.01	69.99 $\pm$ 0.84	74.03 $\pm$ 0.57	6.34
HGAT	90.43 $\pm$ 0.03	91.02 $\pm$ 0.16	88.99 $\pm$ 0.89	89.77 $\pm$ 0.02	90.99 $\pm$ 0.01	89.22 $\pm$ 0.04	71.58 $\pm$ 0.89	72.03 $\pm$ 0.22	7.66
$\kappa$ GCN	<b>92.04<math>\pm</math>0.70</b>	93.33 $\pm$ 0.57	<b>92.45<math>\pm</math>0.85</b>	92.03 $\pm$ 0.63	90.45 $\pm$ 0.88	91.35 $\pm$ 0.60	76.09 $\pm$ 0.76	73.05 $\pm$ 0.71	5.56
QGCN	<b>92.17<math>\pm</math>0.79</b>	92.75 $\pm$ 0.52	92.16 $\pm$ 0.09	91.67 $\pm$ 0.05	<b>91.07<math>\pm</math>0.06</b>	90.98 $\pm$ 0.92	75.44 $\pm$ 0.10	73.89 $\pm$ 0.26	5.65
SelfMGNN	<b>93.12<math>\pm</math>0.04</b>	92.99 $\pm$ 0.91	90.99 $\pm$ 0.17	<b>93.51<math>\pm</math>0.14</b>	91.98 $\pm$ 0.19	<b>95.01<math>\pm</math>0.16</b>	74.51 $\pm$ 0.62	78.99 $\pm$ 0.81	4.28
ChebyNet	88.23 $\pm$ 0.85	89.22 $\pm$ 0.06	86.54 $\pm$ 0.29	90.01 $\pm$ 0.23	88.09 $\pm$ 0.44	92.13 $\pm$ 0.57	73.45 $\pm$ 0.01	79.01 $\pm$ 0.18	7.33
BernNet	86.34 $\pm$ 0.13	87.09 $\pm$ 0.60	85.34 $\pm$ 0.82	87.15 $\pm$ 0.37	87.22 $\pm$ 0.15	91.22 $\pm$ 0.55	<b>77.65<math>\pm</math>0.87</b>	78.34 $\pm$ 0.19	8.12
GPRGNN	91.16 $\pm$ 0.72	93.05 $\pm$ 0.81	92.03 $\pm$ 0.01	91.22 $\pm$ 0.16	89.76 $\pm$ 0.62	92.34 $\pm$ 0.23	76.05 $\pm$ 0.18	<b>80.04<math>\pm</math>0.12</b>	4.96
FiGURe	<b>91.98<math>\pm</math>0.69</b>	<b>94.33<math>\pm</math>0.15</b>	<b>92.67<math>\pm</math>0.83</b>	<b>93.09<math>\pm</math>0.31</b>	90.11 $\pm$ 0.29	<b>95.43<math>\pm</math>0.65</b>	<b>76.99<math>\pm</math>0.16</b>	<b>80.12<math>\pm</math>0.58</b>	3.82
CUSP	<b>95.08<math>\pm</math>0.13</b>	<b>96.88<math>\pm</math>0.65</b>	<b>96.01<math>\pm</math>0.01</b>	<b>97.66<math>\pm</math>0.33</b>	<b>96.04<math>\pm</math>0.38</b>	<b>97.17<math>\pm</math>0.11</b>	<b>81.23<math>\pm</math>0.14</b>	<b>85.23<math>\pm</math>0.05</b>	0
Imp. $\Delta$	1.96	2.55	3.34	4.15	4.06	1.74	3.58	5.11	

Table 8: Performance comparison of CUSP with baselines for LP task (Mean AUC Score  $\pm$  95% confidence interval). **First**, **Second** and **Third** best performing models are highlighted.

Dataset	CUSP	CUSP <sub>euc</sub>	CUSP <sub>lap</sub>	CUSP <sub>enc</sub>	CUSP <sub>pool</sub>	CUSP <sub>fil</sub>
Cora	<b>95.08<math>\pm</math>0.13</b>	92.45 $\pm$ 0.25	93.21 $\pm$ 0.42	93.78 $\pm$ 0.33	92.13 $\pm$ 0.40	93.02 $\pm$ 0.27
Citeseer	<b>96.88<math>\pm</math>0.65</b>	94.03 $\pm$ 0.30	95.34 $\pm$ 0.22	94.08 $\pm$ 0.29	94.01 $\pm$ 0.31	94.56 $\pm$ 0.26
PubMed	<b>96.01<math>\pm</math>0.01</b>	93.52 $\pm$ 0.60	94.81 $\pm$ 0.40	94.92 $\pm$ 0.38	94.11 $\pm$ 0.35	94.16 $\pm$ 0.39
Chameleon	<b>97.66<math>\pm</math>0.33</b>	95.02 $\pm$ 0.44	96.23 $\pm$ 0.52	96.13 $\pm$ 0.28	95.13 $\pm$ 0.40	95.67 $\pm$ 0.57
Actor	<b>96.04<math>\pm</math>0.38</b>	91.45 $\pm$ 0.55	93.99 $\pm$ 0.32	92.81 $\pm$ 0.42	91.98 $\pm$ 0.47	92.13 $\pm$ 0.49
Squirrel	<b>97.17<math>\pm</math>0.11</b>	93.14 $\pm$ 0.22	95.56 $\pm$ 0.37	94.33 $\pm$ 0.43	93.75 $\pm$ 0.32	92.89 $\pm$ 0.14
Texas	<b>81.23<math>\pm</math>0.14</b>	78.45 $\pm$ 0.36	79.88 $\pm$ 0.52	80.12 $\pm$ 0.42	79.23 $\pm$ 0.37	77.81 $\pm$ 0.54
Cornell	<b>85.23<math>\pm</math>0.05</b>	82.89 $\pm$ 0.32	83.91 $\pm$ 0.39	84.23 $\pm$ 0.37	82.31 $\pm$ 0.48	82.45 $\pm$ 0.42
Avg. $\Delta$ Gain	0	<b>3.0437</b>	<b>1.5462</b>	<b>1.8625</b>	<b>2.8312</b>	<b>2.8262</b>

Table 9: Ablation study (LP) results. CUSP<sub>euc</sub> is the Euclidean variant, CUSP<sub>lap</sub> uses the traditional Laplacian, CUSP<sub>enc</sub> gets rid of curvature encoding, CUSP<sub>pool</sub> replaces Cusp pooling with concatenation, and CUSP<sub>fil</sub> uses a single filter instead of a filter bank. **Av.  $\Delta$  Gain** represents the average gain of CUSP over the ablation model in that column, averaged across the different datasets.

CUSP	Cora	Citeseer	PubMed	Chameleon	Actor	Squirrel	Texas	Cornell
$\mathbb{H}^{24} \times \mathbb{S}^{24}$	93.10±0.22	94.33±0.35	95.41±0.25	97.28±0.45	95.25±0.44	96.05±0.51	<b>75.43±0.31</b>	<b>76.95±0.38</b>
$(\mathbb{H}^8)^2 \times (\mathbb{S}^8)^2 \times \mathbb{E}^{16}$	93.00±0.27	95.11±0.28	94.52±0.34	96.66±0.39	95.34±0.38	95.40±0.57	80.95±0.42	84.80±0.43
$\mathbb{H}^8 \times \mathbb{S}^8 \times \mathbb{E}^{32}$	93.20±0.24	93.25±0.30	95.65±0.30	97.11±0.37	94.92±0.40	95.51±0.45	<b>81.23±0.14</b>	<b>85.23±0.05</b>
$\mathbb{H}^{16} \times (\mathbb{S}^{16})^2$	93.50±0.25	94.12±0.33	95.32±0.45	<b>97.66±0.33</b>	95.20±0.34	<b>97.17±0.11</b>	80.30±0.45	<b>78.02±0.47</b>
$(\mathbb{H}^{16})^2 \times \mathbb{E}^{16}$	93.25±0.21	93.05±0.32	<b>96.01±0.01</b>	96.80±0.40	<b>96.04±0.38</b>	96.15±0.55	79.50±0.38	84.60±0.40
$\mathbb{H}^{24} \times \mathbb{E}^{24}$	92.70±0.30	94.71±0.35	95.45±0.42	96.51±0.47	95.00±0.37	<b>88.95±0.53</b>	79.70±0.54	81.20±0.60
$\mathbb{S}^{24} \times \mathbb{E}^{24}$	88.50±0.39	91.23±0.35	89.99±0.44	91.12±0.43	<b>79.90±0.52</b>	91.20±0.51	79.08±0.52	83.75±0.48
$\mathbb{H}^{16} \times \mathbb{S}^{16} \times \mathbb{E}^{16}$	<b>95.08±0.13</b>	<b>96.88±0.65</b>	95.55±0.41	94.44±0.34	95.55±0.35	96.25±0.38	81.07±0.29	84.60±0.30
$(\mathbb{S}^8)^2 \times \mathbb{E}^{32}$	88.05±0.40	90.35±0.42	<b>86.30±0.48</b>	89.34±0.45	93.33±0.54	90.25±0.48	80.10±0.43	82.01±0.55
$(\mathbb{H}^{16})^3$	89.10±0.44	91.01±0.50	93.93±0.55	96.19±0.50	95.25±0.49	92.30±0.52	<b>77.77±0.53</b>	79.44±0.58

Table 10: Performance comparison of CUSP with different manifold signatures for Link Prediction (LP). Best performing *signatures* are in **Bold**, and cases with a large decline in performance because of manifold mismatch are in **Blue**.

Dataset	Signature
Cora	$\mathbb{H}^{16}(-0.21, 0.31) \times \mathbb{S}^{16}(+0.49, 0.38) \times \mathbb{E}^{16}(0, 0.31)$
Citeseer	$\mathbb{H}^{16}(-0.78, 0.29) \times \mathbb{S}^{16}(+0.55, 0.39) \times \mathbb{E}^{16}(0, 0.32)$
PubMed	$\mathbb{H}^{16}(-0.76, 0.56) \times \mathbb{H}^{16}(-0.28, 0.41) \times \mathbb{E}^{16}(0, 0.03)$
Chameleon	$\mathbb{H}^{16}(-0.34, 0.09) \times \mathbb{S}^{16}(+0.71, 0.25) \times \mathbb{S}^{16}(+0.55, 0.34)$
Actor	$\mathbb{H}^{16}(-0.77, 0.17) \times \mathbb{H}^{16}(-0.39, 0.42) \times \mathbb{E}^{16}(0, 0.41)$
Squirrel	$\mathbb{H}^{16}(-0.17, 0.23) \times \mathbb{S}^{16}(+0.54, 0.38) \times \mathbb{S}^{16}(+0.38, 0.39)$
Texas	$\mathbb{H}^8(-0.38, 0.31) \times \mathbb{S}^8(+0.18, 0.19) \times \mathbb{E}^{32}(0, 0.50)$
Cornell	$\mathbb{H}^8(-0.41, 0.19) \times \mathbb{S}^8(+0.09, 0.26) \times \mathbb{E}^{32}(0, 0.55)$

Table 11: Learning results of CUSP on Link Prediction (LP) task for the best performing product signature. Format of entries – *manifold type*<sup>(dim)</sup> (**learn***red* *curvature*, **learn***blue* *manifold weight*).

Dataset	I	Z <sup>(1)</sup>	Z <sup>(2)</sup>	Z <sup>(3)</sup>	Z <sup>(4)</sup>	Z <sup>(5)</sup>	Z <sup>(6)</sup>	Z <sup>(7)</sup>	Z <sup>(8)</sup>	Z <sup>(9)</sup>	Z <sup>(10)</sup>
Cora	<b>0.1125</b>	0.2393	0.0520	0.0504	0.0676	0.0272	<b>0.1531</b>	0.1251	0.0939	0.0095	0.0694
Citeseer	<b>0.2131</b>	0.0150	0.2195	0.0283	<b>0.0922</b>	0.1530	0.0350	0.0320	0.0282	0.1254	0.0582
PubMed	0.0279	0.1793	0.0285	0.0296	0.0563	<b>0.3870</b>	0.0603	0.0324	0.0612	0.0271	<b>0.1104</b>
Chameleon	0.0601	0.1136	0.1694	<b>0.0924</b>	<b>0.1329</b>	0.1605	0.1026	0.0157	0.0475	0.0344	0.0709
Actor	<b>0.1230</b>	0.0321	0.0324	0.1147	0.1287	<b>0.3700</b>	0.0136	0.0389	0.0604	<b>0.0691</b>	0.0172
Squirrel	0.0182	0.0289	<b>0.1056</b>	<b>0.2099</b>	0.0209	0.0355	0.0854	<b>0.2159</b>	0.0475	0.1042	0.1279
Texas	0.0890	<b>0.1983</b>	0.0179	<b>0.4570</b>	0.0035	<b>0.1429</b>	0.0177	0.0087	0.0474	0.0131	0.0046
Cornell	0.0886	<b>0.2026</b>	0.0181	<b>0.4460</b>	0.0034	<b>0.1472</b>	0.0183	0.0089	0.0487	0.0134	0.0048

Table 12: Learned filter weights (Link Prediction) for the top-performing split, distinguishing between homophilic (favoring low-pass filters) and heterophilic (favoring high-pass filters). **First**, **second**, and **third** highest filter weights are highlighted.

#### 7.6.4 ESTIMATING PRODUCT MANIFOLD SIGNATURE

In our model, the mixed-curvature product manifold  $\mathbb{P}^{dc}$  is essential for representing the geometric structure of the data. The curvature configuration needed for each dataset depends on the intrinsic geometry of its graph. To generalize across various datasets, we aim to determine the optimal signature of the product manifold, specifically the proportions of hyperbolic, spherical, and Euclidean components. This estimation is based on analyzing the Ollivier-Ricci curvature (ORC) distribution as a heuristic. See Figures 4 and 5 in Appendix 7.2.1 for the ORC distribution of multiple datasets. For datasets with many positively curved edges, we select a Spherical component, and for those with negatively curved edges, we choose a Hyperbolic component. For example, the curvature distribution of PubMed’s edges in Figure 4 shows two significant peaks around  $-0.45$  and  $+0.25$ . Given this distribution, we opt for Spherical and Hyperbolic components when evaluating CUSP on PubMed. Empirically, the best performance for PubMed is achieved with the signature  $(\mathbb{H}^{16})^2 \times \mathbb{E}^{16}$ . We initialize the curvatures of PubMed’s component manifolds with these prominent values:  $\mathbb{H}$  with  $-0.45$  and  $\mathbb{S}$  with  $+0.25$ . We select two hyperbolic manifolds to capture different curvature ranges.

An overview of this simple idea is provided in Algorithm 1. By systematically analyzing the curvature distribution, our heuristic-based algorithm identifies the manifold signature that best represents the dataset’s underlying geometric structure. We heuristically cluster the curvature distribution and identify the centroid curvatures without altering their order or frequencies. The use of predefined dimensions allows for flexibility based on experimental settings. Since optimal dimension allocations can vary and are complex to analyze, we manually set the dimensions of the component manifolds as a hyperparameter. This ensures fair and uniform comparison across multiple datasets, as different datasets may perform best with different configurations. We do not claim that this algorithm finds the best possible, optimal combination of component manifolds, rather, it *estimates* a potential signature that might be a good fit for a particular dataset.

**Algorithm 1** Product manifold signature estimation and curvature initialisation

**Require:**

- Edge curvature histogram  $\mathcal{C} = \{(\kappa_i, f_i)\}_{i=1}^N$
- Threshold  $\epsilon$  to distinguish between curved and flat regions
- Maximum number of Hyperbolic ( $\mathcal{H}_{\max}$ ) and Spherical ( $\mathcal{S}_{\max}$ ) components.
- Total product manifold dimension  $d_{\mathcal{M}}$
- (Optional) Preferred component manifold dimensions  $d_{(h)}^{\text{pre}}, d_{(s)}^{\text{pre}}, d_{(e)}^{\text{pre}}$

**Ensure:** Product manifold signature  $\mathbb{P}^{d_{\mathcal{M}}} = \times_{q=1}^{\mathcal{Q}} \mathcal{M}_q^{\kappa(q), d(q)}$

- 1: Normalize frequencies:  $f'_i = \frac{f_i}{\sum_{j=1}^N f_j}$
- 2: Construct weighted curvature set:  $\mathcal{C}' = \{(\kappa_i, f'_i)\}_{i=1}^N$
- 3: Determine optimal number of clusters  $K$  using methods like the elbow method, constrained by  $K \leq \mathcal{H}_{\max} + \mathcal{S}_{\max} + 1$  ▷ There can be only 1 Euclidean component
- 4: Cluster  $\mathcal{C}'$  into  $K$  clusters using weighted clustering (e.g., weighted K-means)
- 5: Initialize empty lists  $\mathcal{H}, \mathcal{S}, \mathcal{E}$
- 6: **for** each cluster  $c$  in clusters **do**
- 7:   Compute cluster centroid curvature  $\kappa_c = \frac{\sum_{(\kappa_i, f'_i) \in c} \kappa_i}{|c|}$
- 8:   Compute total frequency weight  $w_c = \sum_{(\kappa_i, f'_i) \in c} f'_i$
- 9:   **if**  $\kappa_c < -\epsilon$  **and**  $|\mathcal{H}| \leq \mathcal{H}_{\max}$  **then** ▷ Negative curvature
- 10:     Assign manifold component:  $\mathcal{M}_q = \mathbb{H}^{\kappa_c}$  ▷ Curvature initialization
- 11:     Add  $(\mathcal{M}_q, w_c)$  to  $\mathcal{H}$
- 12:   **else if**  $\kappa_c > \epsilon$  **and**  $|\mathcal{S}| \leq \mathcal{S}_{\max}$  **then** ▷ Positive curvature
- 13:     Assign manifold component:  $\mathcal{M}_q = \mathbb{S}^{\kappa_c}$  ▷ Curvature initialization
- 14:     Add  $(\mathcal{M}_q, w_c)$  to  $\mathcal{S}$
- 15:   **else**
- 16:     Assign manifold component:  $\mathcal{M}_q = \mathbb{E}$  ▷ Approximate zero curvature, i.e.  $\kappa_c \in [-\epsilon, \epsilon]$
- 17:     Add  $(\mathcal{M}_q, w_c)$  to  $\mathcal{E}$
- 18:   **end if**
- 19: **end for**
- 20: **if** Predefined dimensions  $d_{(h)}^{\text{pre}}, d_{(s)}^{\text{pre}}, d_{(e)}^{\text{pre}}$  are provided **then**
- 21:   Assign dimensions  $d_{(q)}$  to each component  $q$  as per predefined values ▷ Dimension assignment
- 22: **else**
- 23:   Set total number of components  $\mathcal{Q} = |\mathcal{H}| + |\mathcal{S}| + |\mathcal{E}|$  ▷ Dimension assignment
- 24:   Allocate dimensions  $d_{(q)}$  to each component  $q$ :  $d_{(q)} = \left\lfloor d_{\mathcal{M}} \times \frac{w_q}{\sum_{p=1}^{\mathcal{Q}} w_p} \right\rfloor$  ▷ Proportional to weights
- 25:   Adjust  $d_{(q)}$  to ensure  $\sum_{q=1}^{\mathcal{Q}} d_{(q)} = d_{\mathcal{M}}$
- 26: **end if**
- 27: Formulate manifold signature:

$$\mathbb{P}^{d_{\mathcal{M}}} = \left( \times_{h=1}^{|\mathcal{H}|} \mathbb{H}_h^{\kappa(h), d(h)} \right) \times \left( \times_{s=1}^{|\mathcal{S}|} \mathbb{S}_s^{\kappa(s), d(s)} \right) \times \mathbb{E}^{d(e)}$$

## 7.6.5 MORE EXPERIMENTAL SETTINGS

Hyperparameter	Tuning Configurations	Description
$L$	{5, <b>10</b> , 15, 20, 25}	Total number of graph filters.
$\delta$	{0.2, <b>0.5</b> , 0.7}	Neighbourhood weight distribution parameter for ORC
$\alpha$	{0.1, <b>0.3</b> , 0.5, 0.9}	Alpha (initialization) parameter for GPR propagation
$d^c$	{8, <b>16</b> , 32, 64}	Total dimension of curvature embeddings.
$d^{\mathcal{M}}$	{32, <b>48</b> , 64, 128, 256}	Total dimension of the product manifold.
dropout	{0.2, <b>0.3</b> , 0.5}	Dropout rate
epochs	{50, <b>100</b> , 300}	Number of training epochs
lr	{ $1e-4$ , <b><math>4e-3</math></b> , 0.001, 0.01}	Learning rate
weight_decay	{0, $1e-4$ , <b><math>5e-4</math></b> }	Weight decay

Table 13: Hyperparameter configurations used in the experiments for all baselines. Some of the hyperparameters are specific to CUSP. We highlight the final configuration of CUSP for NC in **Red**.