Depth-Adaptive Graph Neural Networks via Learnable Bakry-Émery Curvature

Asela Hevapathige
asela.hevapathige@anu.edu.au
School of Computing
Australian National University
Canberra, Australia

Ahad N. Zehmakan
ahadn.zehmakan@anu.edu.au
School of Computing
Australian National University
Canberra, Australia

Qing Wang
qing.wang@anu.edu.au
School of Computing
Australian National University
Canberra, Australia

Abstract

Graph Neural Networks (GNNs) have demonstrated strong representation learning capabilities for graph-based tasks. Recent advances on GNNs leverage geometric properties, such as curvature, to enhance its representation capabilities by modeling complex connectivity patterns and information flow within graphs. However, most existing approaches focus solely on discrete graph topology, overlooking diffusion dynamics and task-specific dependencies essential for effective learning. To address this, we propose integrating Bakry-Émery curvature, which captures both structural and task-driven aspects of information propagation. We develop an efficient, learnable approximation strategy, making curvature computation scalable for large graphs. Furthermore, we introduce an adaptive depth mechanism that dynamically adjusts message-passing layers per vertex based on its curvature, ensuring efficient propagation. Our theoretical analysis establishes a link between curvature and feature distinctiveness, showing that high-curvature vertices require fewer layers, while low-curvature ones benefit from deeper propagation. Extensive experiments on benchmark datasets validate the effectiveness of our approach, showing consistent performance improvements across diverse graph learning tasks.

1 Introduction

Graph Neural Networks (GNNs) have emerged as a transformative technique for learning graph representations by combining vertex features with structural data. Their strength in capturing relational dependencies has driven their use in various tasks such as vertex classification [65, 67], link prediction [73, 74], and community detection [8, 53]. Typically, GNNs stack multiple message-passing layers, allowing vertices to iteratively aggregate information from their neighbors and improve their representations. This design makes GNNs effective for modeling complex graph structures.

Recently, researchers have leveraged geometric tools, such as curvature [5], to enhance the representational power of GNNs [58, 69]. Originally rooted in differential geometry, curvature measures how a space deviates from flatness by tracking the convergence or divergence of geodesics [5, 43]. When applied to graphs, it captures the connectivity and relational properties of vertices and edges [32, 42], offering a more nuanced structural analysis. Incorporating curvature into GNNs can not only deepen our understanding of local connectivity but also provide insights into information flow, helping mitigate issues like oversmoothing and oversquashing [16, 56]. Among the various forms of curvature, Ricci curvature has gained prominence due to its formulation on discrete spaces, which allows for efficient computation. This efficiency makes it well-suited for analyzing the structural properties of graphs [16, 18, 43, 51].

Despite their promise, current approaches compute curvature as a pre-processing step for GNNs, limiting its integration to decoupled techniques, such as graph rewiring and sampling [16, 31, 36, 41, 58, 69], which do not interact dynamically with the learning process. Consequently, these methods often overlook the interplay between vertex features, structured data, and the evolving dynamics of information diffusion that directly impact downstream tasks. However, integrating curvature dynamically during training presents significant challenges. Traditional curvature measures rely on the static, discrete graph structure, limiting their flexibility in learning contexts. Moreover, dynamic integration requires efficient curvature computation at every training step and the ability to adapt to the continuously changing training process.

In this work, we address these challenges by leveraging Bakry-Émery curvature [9, 38, 46]. Unlike traditional curvature measures, Bakry-Émery curvature captures intrinsic graph structures while accounting for diffusion dynamics, vertex function behavior, and gradient flows. Essentially, it extends Ricci curvature by incorporating a broader range of graph features, offering a more nuanced view of structural properties and information propagation [62]. Notably, although Bakry-Émery curvature is computed locally, its lower bounds provide theoretical guarantees for global properties such as Markov chain convergence rates and functional inequalities [61]. These features make it particularly well-suited for GNN architectures, where understanding the interplay between structure and diffusion can lead to more effective learning strategies.

However, integrating Bakry-Émery curvature into the GNN architecture is non-trivial. One key difficulty arises from its definition [2], which requires evaluating curvature over the entire function space, a computationally intractable task for large, complex graphs. To

overcome this, we propose a learnable strategy that selects a task-specific subset of functions, enabling efficient curvature estimation by focusing on the most relevant structural and diffusion characteristics. Moreover, we introduce an adaptive mechanism that leverages the estimated vertex curvature to dynamically adjust the depth of message-passing layers for each vertex. This approach not only enhances the feature distinctiveness of GNN architectures but also captures finer structural variations within the graph.

To summarize, our main contributions are as follows:

- **A New Perspective on Curvature in GNNs:** We propose Bakry-Émery curvature as a geometric tool for measuring vertex curvature in GNNs, capturing both structural properties and diffusion dynamics of graphs.
- **Efficient Curvature Approximation:** We develop a learnable strategy that selects a task-specific subset of functions, enabling efficient estimation of Bakry-Émery curvature on large-scale graphs.
- **Depth-Adaptive Mechanism for GNNs:** We introduce an adaptive mechanism that uses estimated Bakry-Émery curvature to dynamically determine the message-passing depth for each vertex, thereby enhancing the feature distinctiveness of existing GNN architectures.
- **Theoretical Insights:** We theoretically link Bakry-Émery curvature to feature distinctiveness in GNNs, showing that high-curvature vertices (fast diffusion, short mixing times) need fewer layers for distinct representations, while low-curvature vertices (slow diffusion, long mixing times) benefit from deeper architectures.
- **Enhanced GNN Performance:** Extensive experiments show that integrating our depth-adaptive mechanism consistently improves the performance of existing GNN architectures across a range of downstream tasks.

2 Related Work

2.1 Curvature-based GNNs

Curvature, originally defined on smooth manifolds via geodesics, has been successfully extended to discrete structures like graphs [29, 44]. Recent work in GNNs has leveraged various notions of curvature, most notably discrete Ricci curvature, to capture structural relationships in graphs. For instance, Ye et al. [69] employed discrete Ricci curvature to quantify the relationship between the neighborhoods of vertex pairs, using these values to define edge weights and enhance message passing. Subsequent studies [16, 36, 41, 58] have further explored how curvature influences common GNN challenges: oversmoothing (where vertex representations become overly similar, typically linked to regions of positive curvature) and oversquashing (where long-range information is inadequately propagated, often associated with negative curvature). Building on these insights, Fesser and Weber [16], Nguyen et al. [41], Topping et al. [58] introduced graph rewiring strategies to improve information flow, while Liu et al. [36] proposed an edge sampling mechanism that leverages curvature to alleviate these issues. However, a common limitation of these approaches is their treatment of curvature as a fixed, intrinsic property derived solely from the graph topology (i.e., precomputed before training), which can restrict adaptability and generalization in learning scenarios. More recently, Chen et al. [7] proposed a curvature-based diffusion mechanism that iteratively updates edge curvature, effectively treating curvature as a learnable parameter. Although this approach introduces greater flexibility, its lack of a rigorous theoretical foundation for curvature learning raises concerns about potential instability and overfitting, as the model must concurrently infer both the curvature and the underlying graph structure.

Our work overcomes these limitations by adaptively learning curvature tailored to the downstream task, enhancing generalizability. We provide a theoretically grounded framework that captures both graph geometry and the behavior of information propagation functions over the graph, accounting for the graph's topology and diffusion dynamics, thereby offering a principled measure of information propagation to guide the learning process. Unlike previous methods that define curvature at the edge level, we define curvature at the vertex level, offering a more nuanced representation of the graph structure.

2.2 GNNs with Adaptive Architectures

GNNs with adaptive architectures learn key components of their architecture during training, allowing them to optimize both structure and operations based on the graph's properties and the downstream task. Unlike conventional GNNs with fixed designs, these models incorporate learnable components that dynamically adjust aggregation strategies, assign importance to vertices/edges, and refine connectivity patterns. Numerous studies have explored adaptive behaviors of GNNs in the literature. For example, sampling-based methods selectively extract graph structures to improve computational efficiency and representation capabilities [30, 71, 75]. Similarly, several studies focus on defining adaptive neighborhoods for aggregation at each vertex by utilizing learnable mechanisms that are governed by the ground truth values of the downstream task [22, 50]. Attention-based GNNs [6, 60] adjust edge weights adaptively by learning attention coefficients that reflect the relative importance of neighboring vertices during the learning process. Additionally, several studies have proposed dynamic message-passing paradigms that can adaptively determine vertex message-passing operations, such as message filtering and the direction of information propagation [12, 17].

Our approach differs fundamentally from existing adaptive GNN methods by introducing an adaptive layer depth mechanism guided by learnable vertex curvatures. Specifically, we define per-vertex adaptive depth, allowing the model to adjust the number of message-passing steps for each vertex according to its curvature.

3 Background

Let $G = (V, E, w)$ be an undirected weighted graph, where V is the set of vertices, E is the set of edges, and $w : E \rightarrow \mathbb{R}_+$ is the edge weight function. For any vertex $x \in V$, we denote by $N(x)$ the set of vertices adjacent to x.

3.1 Graph Neural Networks

Graph Neural Networks (GNNs) [64] extend deep learning to non-Euclidean domains by learning directly from graph-structured data. The typical GNN architecture follows a message-passing framework, consisting of two main operations:

$$\text{Aggregation:} \quad m_x^{(t)} = \text{AGG}\left(\{\!\{h_y^{(t-1)} \mid y \in N(x)\}\!\}\right),$$

$$\text{Update:} \quad h_x^{(t)} = \text{UPD}\left(h_x^{(t-1)}, m_x^{(t)}\right).$$

For each vertex x at layer t, the aggregated message $m_x^{(t)}$ is computed from the features of its neighbors $N(x)$ using the aggregation function AGG. The vertex representation $h_x^{(t)}$ is then updated by combining the previous state $h_x^{(t-1)}$ with the aggregated message via the update function UPD. Different choices for these functions yield various GNN architectures [23, 28, 60, 68].

In this work, we leverage Bakry-Émery curvature to characterize underlying diffusion patterns and structural properties of a graph. We incorporate curvature into existing GNN architectures in a learnable manner through a depth-adaptive layer mechanism to enhance their representational capacity.

3.2 Bakry-Émery Curvature Formulation

Bakry–Émery curvature provides a way to capture the local geometric behavior of functions on a graph [2]. We begin by defining a few operators that are fundamental to this concept.

For a function $f : V \to \mathbb{R}$, its weighted Laplacian at x is defined as

$$\Delta f(x) = \sum_{y \in N(x)} w(x,y)(f(y) - f(x)). \quad (1)$$

This operator captures the weighted difference between $f(x)$ and the values of f at its neighbors.

To formalize the Bakry–Émery curvature, we introduce two operators that capture the local behavior of functions on the graph. The first operator, $\Gamma(f,f)(x)$, analogous to the squared gradient, quantifies the local variability of f at x relative to its neighbors:

$$\Gamma(f,f)(x) = \frac{1}{2}\left(\sum_{y \in N(x)} w(x,y)(f(y) - f(x))^2\right) \quad (2)$$

The second operator, $\Gamma_2(f,f)(x)$, quantifies the convexity of f at x as follows:

$$\Gamma_2(f,f)(x) = \frac{1}{2}\Delta\Gamma(f,f)(x) - \Gamma(f, \Delta f)(x) \quad (3)$$

which can be equivalently expanded as,

$$\Gamma_2(f,f)(x) = \frac{1}{2}\left(\sum_{y \in N(x)} w(x,y)\big(\Gamma(f,f)(y) - \Gamma(f,f)(x)\big)\right)$$
$$- \sum_{y \in N(x)} w(x,y)(f(y) - f(x))\big(\Delta f(y) - \Delta f(x)\big).$$

The Bakry–Émery curvature $\kappa(x)$ at vertex x is defined as the largest constant such that the curvature–dimension inequality

$$\Gamma_2(f,f)(x) \geq \kappa(x)\,\Gamma(f,f)(x)$$

holds for all functions $f : V \to \mathbb{R}$. Equivalently, $\kappa(x)$ represents the tightest lower bound on the ratio

$$\frac{\Gamma_2(f,f)(x)}{\Gamma(f,f)(x)},$$

for all functions f with $\Gamma(f,f)(x) \neq 0$. This lower bound reflects the local geometric and functional properties of the graph at the vertex x.

4 Curvature and Information Propagation

In this section, we examine the influence of Bakry–Émery curvature on local information diffusion in a graph. In particular, we establish a relationship between curvature and the local mixing time, which quantifies the rate at which information equilibrates around a vertex. The proof is included in the appendix.

Given a vertex $x \in V$ with Bakry–Émery curvature $\kappa(x)$, we define the local mixing time $\tau_x(\epsilon)$ for any tolerance $\epsilon > 0$ as

$$\tau_x(\epsilon) = \inf\left\{t > 0 : |\nabla f_t|^2(x) \leq \epsilon\,|\nabla f_0|^2(x) \text{ for all } f_0 \in \ell^2(V)\right\}$$

Here, $\ell^2(V)$ denotes the space of square-summable functions on V, and $f_t = e^{-tL}f_0$ describes the evolution of f_0 under the heat semigroup associated with the Laplacian L. The operator e^{-tL} is defined via the matrix exponential.

The local gradient at x is defined by

$$|\nabla f|(x) \coloneqq \sqrt{\Gamma(f,f)(x)}.$$

THEOREM 4.1 (MIXING TIME BOUND). *Let $G = (V, E, w)$ be an undirected, weighted graph with bounded degree and Laplacian L. If the local curvature-dimension inequality holds for every $f \in \ell^2(V)$ at vertex x, then for any $\epsilon \in (0, 1)$, the local mixing time satisfies*

$$\tau_x(\epsilon) \leq \frac{\log(1/\epsilon)}{\kappa(x)}.$$

This theorem indicates that higher Bakry–Émery curvature at a vertex accelerates the local decay of gradients, thereby enabling faster information diffusion within its neighborhood. In essence, vertices with high curvature act as efficient hubs, quickly equilibrating information and promoting rapid propagation throughout the graph. Conversely, low-curvature vertices, which are less effective at mixing, lead to slower information spread. Since the mixing time is inversely proportional to curvature, enhancing curvature in the network can be seen as a mechanism for improving diffusion efficiency.

5 Depth-Adaptive GNNs

We introduce a novel mechanism that leverages Bakry–Émery curvature to adaptively control the message-passing depth of graph neural networks (GNNs). In our framework, vertices with higher curvature (which typically indicate rapid local diffusion) terminate message passing earlier, while vertices with lower curvature continue aggregating messages to capture a broader neighborhood. The theoretical justification for this design choice is provided in Section 6.

5.1 Adaptive Layer Depth Mechansim

Let $t \in \mathbb{N}$ denote the iteration index in the message-passing process. To assign a stopping depth $T(x) \in \mathbb{N}$ to each vertex x, we rank the vertices based on their Bakry–Émery curvature $\kappa(x)$. For a given threshold $k\%$, we define

$$T(x) \coloneqq \min\left\{t \in \mathbb{N} \,\middle|\, \frac{1}{|V|}\sum_{y \in V} \mathbb{I}\big(\kappa(y) \geq \kappa(x)\big) \leq \frac{k\,t}{100}\right\},$$

where $\mathbb{I}(\cdot)$ is the indicator function which is 1 if $\kappa(y) \geq \kappa(x)$ and 0 otherwise. This ensures that a vertex x is assigned a stopping depth $T(x) = t$ when the proportion of vertices with curvature at least $\kappa(x)$ falls below the threshold $\frac{k\,t}{100}$.
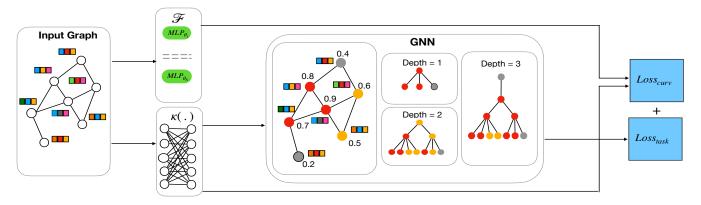
Figure 1: High-level overview of the proposed model: Vertices are clustered based on learned curvature, where message-passing depth is adaptively adjusted according to curvature. Both the curvature functions and the GNN are jointly optimized via a loss function.

During message passing, for each vertex x and iteration $t \leq T(x)$, we update its embedding by aggregating messages from its neighbors. To incorporate the adaptive depth, the message from a neighbor y is taken from the layer $\gamma = \min\{t-1, T(y)\}$. Formally, the update equations are:

$$m_x^{(t)} = \text{AGG}\Big(\{\!\{h_y^{(\min\{t-1, T(y)\})} \mid y \in N(x)\}\!\}\Big), \qquad (4)$$

$$h_x^{(t)} = \text{UPD}\Big(h_x^{(t)}, m_x^{(t-1)}\Big). \qquad (5)$$

After $t = T(x)$ iterations, the final representation $h_x^{(T(x))}$ is fed into a task-specific module (e.g., a classifier or regressor) C:

$$\hat{y}_x = C\big(h_x^{(T(x))}\big)$$

where $\hat{y}_x$ represents the predicted output for vertex x.

This adaptive scheme is model-agnostic and can be integrated into existing GNN architectures such as GIN [68], GCN [28], GAT [60], and GraphSAGE [23]. The high-level architecture of our approach is depicted in Figure 1.

5.2　Learning to Estimate Curvature

Accurate estimation of $\kappa(x)$ is essential for determining the stopping depth $T(x)$. We cast curvature estimation as an optimization problem that enforces the Bakry–Émery curvature-dimension inequality. Let $\hat{\kappa}(x)$ denote the estimated curvature of vertex x. For any function $f : V \to \mathbb{R}$, we define the penalty

$$\mathcal{L}(\hat{\kappa}(x), f) := \max\big\{0, \ \hat{\kappa}(x)\,\Gamma(f,f)(x) - \Gamma_2(f,f)(x)\big\}.$$

This penalty quantifies the violation of the inequality

$$\Gamma_2(f,f)(x) \geq \hat{\kappa}(x)\,\Gamma(f,f)(x).$$

When the inequality holds at vertex x for f, we have $\mathcal{L}(\hat{\kappa}(x), f) = 0$; otherwise, the penalty is positive. Ideally, the true curvature $\kappa_{\text{true}}(x)$ is the largest value such that

$$\mathcal{L}(\kappa_{\text{true}}(x), f) = 0 \quad \text{for all } f : V \to \mathbb{R}.$$

Since optimizing over the entire function space is intractable, we restrict our search to a finite, parameterized set $\mathcal{F}$. In practice, we employ a Multi-Layer Perceptron (MLP) as a flexible function approximator. Let $f_\theta : V \to \mathbb{R}$ denote the function represented by

an MLP with parameters θ. By sampling parameter configurations $\{\theta_i\}_{i=1}^N$, we obtain a candidate set

$$\mathcal{F} = \{f_{\theta_1}, f_{\theta_2}, \dots, f_{\theta_N}\}.$$

We then approximate the estimated curvature by

$$\hat{\kappa}(x) = \sup_{\kappa \in \mathbb{R}} \inf_{f \in \mathcal{F}} \mathcal{L}(\kappa, f).$$

This optimization process seeks the maximum $\hat{\kappa}(x)$ such that the curvature-dimension inequality is "nearly" satisfied for all functions $f \in \mathcal{F}$.

We highlight two key properties of our curvature estimation approach:

(1) **Upper bound:** Since $\mathcal{F}$ is a proper subset of the space of all functions $f : V \to \mathbb{R}$, the restricted optimization can only overestimate the true curvature. Formally, if

$$\kappa_{\text{true}}(x) = \sup\big\{\kappa \in \mathbb{R} : \mathcal{L}(\kappa, f) = 0 \text{ for all } f : V \to \mathbb{R}\big\},$$

then

$$\hat{\kappa}(x) = \sup_{\kappa \in \mathbb{R}} \inf_{f \in \mathcal{F}} \mathcal{L}(\kappa, f) \geq \kappa_{\text{true}}(x).$$

(2) **Smoothness:** In order for the differential operators Γ and Γ_2 to be well-defined, the candidate functions in $\mathcal{F}$ must be smooth [4]. However, standard MLPs do not inherently guarantee smooth approximations, as many common activation functions (e.g., ReLU) lack smoothness. To ensure smooth outputs, we employ an MLP with C^∞ activation functions (e.g., the sigmoid function in our implementation), thereby guaranteeing that each function $f_\theta \in \mathcal{F}$ is infinitely differentiable.

This approach leverages the expressive power of neural networks to approximate a rich subset of functions, facilitating a robust, task-specific estimation of vertex curvature.

5.3　GNN Training Loss Function

We jointly optimize the GNN for the downstream task and for compliance with the curvature-dimension inequality by combining two loss terms: the task loss and the curvature loss.

Let $\mathcal{L}_{\text{task}}$ denote the loss for the downstream task (e.g., cross-entropy for classification or mean squared error for regression). To

Methods	Cora	Citeseer	Pubmed	Actor	Squirrel	Cornell	Wisconsin	Texas	ogbn-arxiv
GCN	87.28 ±1.26	76.68 ±1.64	87.38 ±0.66	30.26 ±0.79	36.89 ±1.34	57.03 ±4.67	59.80 ±6.99	59.46 ±5.25	71.74 ±0.29
GCN+BEC	88.50 ±1.32	80.43 ±0.88	88.55 ±0.68	31.66 ±0.58	37.82 ±0.60	61.62 ±4.64	69.41 ±2.85	75.68 ±2.18	71.91 ±0.17
Δ ↑	+1.22	+3.75	+1.17	+1.40	+0.93	+4.59	+9.61	+16.22	+0.17
GAT	82.68 ±1.80	75.46 ±1.72	84.68 ±0.44	26.28 ±1.73	30.62 ±2.11	58.92 ±3.32	55.29 ±8.71	58.38 ±4.45	71.54 ±0.30
GAT+BEC	85.39 ±0.69	76.12 ±3.23	86.71 ±0.46	31.22 ±0.56	31.61 ±0.47	61.62 ±5.64	63.23 ±3.28	71.56 ±5.56	71.84 ±0.31
Δ ↑	+2.71	+0.66	+2.03	+4.94	+0.99	+2.70	+7.94	+13.18	+0.30
GraphSAGE	86.90 ±1.04	76.04 ±1.30	88.45 ±0.50	34.23 ±0.99	41.61 ±0.74	75.95 ±5.01	81.18 ±5.56	82.43 ±6.14	71.49 ±0.27
GraphSAGE+BEC	87.08 ±0.42	78.01 ±1.33	89.00 ±0.31	35.56 ±0.87	44.53 ±0.81	81.08 ±3.82	88.97 ±1.88	89.41 ±3.06	71.83 ±0.14
Δ ↑	+0.18	+1.97	+0.55	+1.33	+2.92	+5.13	+7.79	+6.98	+0.34
SGC	84.97 ±1.95	75.66 ±1.37	87.15 ±0.47	25.83 ±1.09	41.78 ±2.88	55.41 ±5.29	57.84 ±4.83	58.11 ±6.27	68.74 ±0.12
SGC+BEC	85.49 ±0.65	78.32 ±0.66	87.94 ±0.23	27.97 ±2.18	42.03 ±0.62	65.13 ±3.29	63.97 ±9.30	69.80 ±1.79	71.39 ±0.30
Δ ↑	+0.52	+2.66	+0.79	+2.14	+0.25	+9.72	+6.13	+11.69	+2.65
MixHop	87.61 ±0.85	76.26 ±1.33	85.31 ±0.61	32.22 ±2.34	43.80 ±1.48	73.51 ±6.34	75.88 ±4.90	77.84 ±7.73	70.83 ±0.30
MixHop+BEC	87.89 ±0.67	76.79 ±0.95	88.18 ±0.37	36.91 ±0.93	43.97 ±0.67	78.37 ±2.70	93.82 ±1.10	88.23 ±1.75	72.04 ±0.26
Δ ↑	+0.28	+0.53	+2.87	+4.69	+0.17	+4.86	+17.94	+10.39	+1.21
DIR-GNN	82.89 ±0.44	76.55 ±0.72	88.84 ±0.13	30.12 ±0.65	42.50 ±0.80	80.00 ±3.60	79.60 ±3.80	81.40 ±2.40	70.72 ±0.26
DIR-GNN+BEC	84.12 ±0.76	77.67 ±1.08	89.40 ±0.19	32.46 ±0.43	46.19 ±1.26	85.94 ±3.78	89.55 ±1.22	87.84 ±3.01	71.55 ±0.26
Δ ↑	+1.23	+1.12	+0.56	+2.34	+3.69	+5.94	+9.95	+6.44	+0.83

Table 1: Vertex classification accuracy ± std (%). The baseline results are sourced from [6, 24, 55, 76].

enforce the curvature-dimension inequality, we define the curvature loss as

$$\mathcal{L}_{\text{curv}} = \sum_{x \in V} \left(\sum_{f \in \mathcal{F}} \mathcal{L}\big(\hat{\kappa}(x), f\big) - \lambda \, \hat{\kappa}(x) \right).$$

Here, λ is a regularization constant controlling the tradeoff between maximizing $\hat{\kappa}(x)$ and ensuring the curvature-dimension inequality. The overall training loss is then defined as

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \mathcal{L}_{\text{curv}}.$$

optimizes GNN performance while enforcing adherence to the graph's geometric structure via curvature.

6 Theoretical Analysis

To gain deeper insight into the computational and representational properties of our method, we first analyze its complexity, then explore the connection between feature distinctiveness and curvature.

6.1 Complexity Analysis

We first analyze the time and space complexity of our optimization process. For each vertex $x \in V$, let d_x denote its degree, and let $d_{\max}$ denote the maximum vertex degree in the graph. The computational cost of each operation depends on the local structure of the graph, particularly the degree distribution.

The time complexity of Equation (1), the weighted Laplacian $\Delta f(x)$, and Equation (2), the squared gradient operator $\Gamma(f, f)(x)$, is $O(d_x)$. In contrast, computing the convexity operator $\Gamma_2(f, f)(x)$ (Equation (3)) requires $O(d_x^2)$ operations per vertex. Thus, the overall computational cost over all vertices is $O(\sum_{x \in V} d_x^2)$, which can be upper-bounded by $O(|V| \cdot d_{\max}^2)$. In our approach, the optimization is performed over a finite set $\mathcal{F}$ of candidate functions, with $|\mathcal{F}| = N$. Consequently, the total time complexity becomes

$$O\big(N \cdot |V| \cdot d_{\max}^2\big).$$

Since real-world graphs are typically sparse (i.e., $d_{\max} \ll |V|$) and N is small (in our experiments, $N \leq 5$), the overall time complexity is manageable. The space complexity is $O(|V| + |E|)$, accounting for the storage of vertex and edge data.

We further analyze empirical runtime and parameter complexity in Section 7.3.

6.2 Feature Distinctiveness and Curvature

We establish a connection between a vertex's feature distinctiveness in a GNN and its Bakry–Émery curvature. Here, feature distinctiveness is quantified by the variation in vertex features across layers. In particular, we show that the rate at which feature distinctiveness decays is governed by the Bakry–Émery curvature, reflecting how quickly information diffuses over the graph. This analysis assumes that vertex features evolve approximately according to heat flow driven by the graph Laplacian.

THEOREM 6.1 (FEATURE DECAY BOUND). *Let $G = (V, E, w)$ be an undirected, weighted graph with bounded degree. Consider a GNN with l layers, where each layer approximates the heat flow for a time step Δt. For a vertex $x \in V$ with Bakry-Émery curvature $\kappa(x)$, we define the feature distinctiveness $D(x, l)$ after l layers as*

$$D(x, l) = \frac{|\nabla f_l|^2(x)}{|\nabla f_0|^2(x)},$$

where f_l represents the vertex features at layer l. Then, the following holds:

$$D(x, l) \leq e^{-\kappa(x) l \Delta t}.$$

For any $\epsilon > 0$, to maintain $D(x, l) \geq \epsilon$, it suffices that the number of layers satisfy:

$$l \leq \frac{\log(1/\epsilon)}{\kappa(x) \Delta t}.$$

Methods	Jazz		Cora-ML		Network Science		Power Grid	
	IC	LT	IC	LT	IC	LT	IC	LT
GCN	$0.233_{\pm 0.010}$	$0.199_{\pm 0.006}$	$0.277_{\pm 0.007}$	$0.255_{\pm 0.008}$	$0.270_{\pm 0.019}$	$0.190_{\pm 0.012}$	$0.313_{\pm 0.024}$	$0.335_{\pm 0.023}$
GCN+BEC	$0.202_{\pm 0.007}$	$0.124_{\pm 0.002}$	$0.258_{\pm 0.006}$	$0.194_{\pm 0.010}$	$0.233_{\pm 0.010}$	$0.123_{\pm 0.026}$	$0.329_{\pm 0.019}$	$0.298_{\pm 0.039}$
Δ ↑	+0.031	+0.075	+0.019	+0.061	+0.037	+0.067	-0.016	+0.037
GAT	$0.342_{\pm 0.005}$	$0.156_{\pm 0.100}$	$0.352_{\pm 0.004}$	$0.192_{\pm 0.010}$	$0.274_{\pm 0.002}$	$0.114_{\pm 0.008}$	$0.331_{\pm 0.002}$	$0.280_{\pm 0.015}$
GAT+BEC	$0.238_{\pm 0.035}$	$0.123_{\pm 0.002}$	$0.269_{\pm 0.014}$	$0.184_{\pm 0.015}$	$0.233_{\pm 0.016}$	$0.126_{\pm 0.010}$	$0.291_{\pm 0.016}$	$0.277_{\pm 0.013}$
Δ ↑	+0.104	+0.033	+0.083	+0.008	+0.041	-0.012	+0.040	+0.003
GraphSAGE	$0.201_{\pm 0.028}$	$0.120_{\pm 0.004}$	$0.255_{\pm 0.010}$	$0.203_{\pm 0.019}$	$0.241_{\pm 0.010}$	$0.112_{\pm 0.005}$	$0.313_{\pm 0.024}$	$0.341_{\pm 0.018}$
GraphSAGE+BEC	$0.190_{\pm 0.015}$	$0.060_{\pm 0.010}$	$0.246_{\pm 0.008}$	$0.176_{\pm 0.008}$	$0.231_{\pm 0.010}$	$0.073_{\pm 0.012}$	$0.302_{\pm 0.028}$	$0.205_{\pm 0.013}$
Δ ↑	+0.011	+0.060	+0.009	+0.027	+0.010	+0.039	+0.011	+0.136

Table 2: Influence estimation performance MAE ± std under IC and LT diffusion models.

In a nutshell, the higher curvature vertices, with stronger local connectivity, lose feature distinctiveness more quickly in a GNN due to faster information propagation. Therefore, fewer layers are required to maintain a given level of distinctiveness ϵ for high curvature vertices. In contrast, low curvature vertices have weaker local connections, causing feature distinctiveness to decay more slowly. To preserve ϵ for low curvature vertices, more layers are needed. Thus, the number of layers required to maintain feature distinctiveness tends to decrease with increasing vertex curvature.

7 Experiments

We conduct experiments on four widely used benchmark tasks: vertex classification, vertex regression, graph classification, and graph regression, utilizing a total of 21 datasets and comparing against 10 baselines to evaluate the effectiveness of our approach.

7.1 Experimental Setup

7.1.1 Datasets. For vertex classification, we use three widely adopted graphs exhibiting strong homophily: Cora, PubMed, and Citeseer [40, 52], as well as five datasets with heterophily: Texas, Wisconsin, Actor, Squirrel, and Cornell [49, 57]. Additionally, we include a large-scale dataset from the Open Graph Benchmark, ogbn-arxiv [25]. For vertex regression, we utilize four real-world datasets (Jazz, Network Science, Cora-ML, and Power Grid) [37, 48]. For graph classification, we use six small-scale real-world datasets from TU Datasets (MUTAG, PTC-MR, COX2, BZR, PROTEINS, and IMDB-BINARY) [39], as well as a large-scale molecular dataset from the Open Graph Benchmark, ogbg-moltox21 [25]. Finally, we employ the ZINC 12k dataset [11] for graph regression.

7.1.2 Baselines. For vertex classification and regression baselines, we use three classical GNNs—GCN [28], GAT [60], and GraphSAGE [23]—and for vertex classification, we further employ three enhanced GNNs—SGC [64], MixHop [1], and DIR-GNN [47]. For graph classification and regression baselines, we select two GNNs whose expressive power is known to be upper-bounded by the 1-dimensional Weisfeiler-Lehman (1-WL) algorithm [72]: GIN [68] and GCN [28]. Additionally, we include three GNNs designed to possess expressive power beyond 1-WL: ID-GNN [70], GraphSNN [63], and NC-GNN [35].

7.1.3 Evaluation settings. For vertex classification, we use the feature vectors and class labels with 10 random splits for all datasets

except the OGB dataset. In these splits, 48% of the vertices are assigned for training, 32% for validation, and 20% for testing, following Pei et al. [45]. For the OGB dataset, we adopt the setup described by Hu et al. [25]. For the vertex regression task, we focus on influence estimation [34, 66], a key problem in network science, where the goal is to predict each vertex's influence based on graph structure and diffusion dynamics. We consider the Linear Threshold (LT) [21] and Independent Cascade (IC) [20] models, which capture different information spread mechanisms, with an initial activation set comprising 10% of vertices. Following Ling et al. [34], we use their data splits and adopt 10-fold cross-validation, as in Errica et al. [13]. In graph classification, we adopt the setup outlined by Hu et al. [25] for the OGB dataset. For the TU datasets, we follow the experimental setup provided by Feng et al. [15], reporting the mean and standard deviation of the best accuracy across 10 test folds. For graph regression, we adhere to the setup provided by Dwivedi et al. [11]. We report baseline results from previous works using the same experimental setup, where available. If such results are unavailable, we generate baseline results by adopting the hyperparameter configurations specified in the original papers.

Since the benchmark graphs are unweighted and the Bakry-Émery curvature operators (Equations (1), (2), and (3)) require edge weights, we employ learnable parameters for these weights. Additionally, the functions used to evaluate the Bakry-Émery curvature inequality take vertex features as inputs.

Comprehensive details regarding downstream tasks, baselines, dataset statistics, and model hyperparameters can be found in the Appendix.

7.2 Main Results

We use GNN+BEC to represent the GNN model incorporating our Bakry-Émery curvature-based adaptive layer depth mechanism, and Δ ↑ indicates the performance improvement of our approach compared to the baseline.

7.2.1 Vertex Classification. Table 1 presents the results for the vertex classification task. Our approach consistently outperforms the baselines in both homophilic and heterophilic settings. Notably, our adaptive layer mechanism achieves greater performance improvements on heterophilic graphs compared to homophilic ones. We attribute this to the ability of our method to learn data-driven curvature values that leverage heterophilic label patterns, effectively

Methods	MUTAG	PTC-MR	IMDB-BINARY	COX2	BZR	PROTEINS	ogbg-moltox21	ZINC
GIN	92.8 ±5.9	65.6 ±6.5	78.1 ±3.5	88.9 ±2.3	91.1 ±3.4	78.8 ±4.1	74.9 ±0.5	0.387 ±0.015
GIN+BEC	96.1 ±3.6	72.9 ±5.7	80.8 ±3.3	89.3 ±3.1	92.4 ±3.6	79.1 ±3.7	75.7 ±0.7	0.308 ±0.009
$\Delta \uparrow$	+3.3	+7.3	+2.7	+0.4	+1.3	+0.3	+0.8	+0.079
GCN	92.2 ±4.4	68.8 ±6.2	79.8 ±2.3	88.5 ±3.8	92.6 ±4.8	78.8 ±3.9	75.3 ±0.7	0.459 ±0.006
GCN+BEC	95.0 ±3.0	70.6 ±5.1	79.9 ±3.4	89.3 ±2.8	93.6 ±3.8	79.6 ±4.4	75.5 ±0.5	0.424 ±0.049
$\Delta \uparrow$	+2.8	+1.8	+0.1	+0.8	+1.0	+0.8	+0.2	+0.035
ID-GNN	97.8 ±3.7	74.4 ±4.2	79.3 ±2.9	87.8 ±3.1	93.3 ±4.8	78.1 ±3.9	74.7 ±0.5	0.366 ±0.014
ID-GNN+BEC	98.3 ±3.6	75.6 ±4.0	81.5 ±2.4	89.3 ±3.1	93.8 ±4.5	78.6 ±3.8	75.4 ±0.7	0.350 ±0.013
$\Delta \uparrow$	+0.5	+0.8	+2.2	+1.5	+0.5	+0.5	+0.7	+0.016
GraphSNN	94.7 ±1.9	70.6 ±3.1	78.5 ±2.3	86.3 ±3.3	91.1 ±3.0	78.4 ±2.7	75.5 ±1.1	0.297 ±0.004
GraphSNN+BEC	96.1 ±2.5	72.9 ±7.4	79.4 ±2.7	88.9 ±2.0	92.1 ±4.9	79.2 ±3.9	75.3 ±0.7	0.265 ±0.005
$\Delta \uparrow$	+1.4	+2.3	+0.9	+2.6	+1.0	+0.8	-0.2	+0.032
NC-GNN	92.8 ±5.0	71.8 ±6.2	78.4 ±4.0	88.4 ±3.3	92.6 ±4.3	78.4 ±3.1	75.1 ±0.4	0.448 ±0.095
NC-GNN+BEC	96.0 ±2.4	75.0 ±4.6	79.6 ±3.6	88.9 ±2.4	93.6 ±4.9	79.6 ±2.9	75.6 ±0.5	0.356 ±0.017
$\Delta \uparrow$	+3.2	+3.2	+1.2	+0.5	+1.0	+1.2	+0.5	+0.092

Table 3: Graph classification accuracy ± std (%) for TU datasets, ROC ± std (%) for the OGB dataset, and graph regression MAE ± std for ZINC dataset, with baseline results sourced from Wijesinghe and Wang [63], Feng et al. [15], and Hu et al. [25].

capturing the complex and diverse relationships between vertices with dissimilar labels, which leads to enhanced aggregation and feature extraction. Additionally, our approach improves the performance of GNNs designed with inductive biases for heterophilic graph settings, such as MixHop, and DIR-GNN. This demonstrates that our method complements these models by capturing additional structural and label-dependent information that might otherwise be overlooked.

We also observe that our depth-adaptive mechanism significantly reduces the standard deviation of accuracy across baseline models, enhancing stability and consistency. This reduced variance ensures reliable and repeatable results, essential for real-world applications.

7.2.2 Vertex Regression. Table 2 presents the results for the influence estimation task, formulated as a vertex regression problem. The results indicate that our approach significantly enhances baseline performance in predicting influence by effectively adapting to various underlying diffusion processes. By dynamically adjusting curvature based on the propagation mechanisms of different diffusion models, our method enhances generalization of the baseline GNN models across diverse network conditions.

7.2.3 Graph Classification. We present graph classification results in Table 3. These results show that our adaptive layer mechanism achieves notable to moderate improvements over baseline methods across all datasets. Significant gains are observed with the GIN model, which is commonly regarded as the standard baseline for graph classification tasks. Moreover, our approach enhances GNNs with higher expressive power than 1-WL by improving feature distinctiveness, thus offering a more effective mechanism for distinguishing between graph structures.

7.2.4 Graph Regression. The regression results for ZINC 12K dataset is depicted in Table 3. Similar to the graph classification, the proposed adaptive layer mechanism consistently improves performance over baseline methods. These results underscore the effectiveness of our approach in enhancing feature representation and regularization, thereby improving the overall performance of graph

regression tasks. Although vertex-specific, our curvature provides insights into the global graph structure, enhancing performance in tasks that depend on holistic properties of graphs.

7.3 Ablation Studies

In this section, we conduct experimental analysis to assess different aspects of our work.

7.3.1 Hyper-Parameter Analysis. In this experiment, we analyze the model's sensitivity to its two primary hyper-parameters. Using GraphSAGE as the base model, we evaluate vertex classification performance on two datasets. Figure 2 (a) illustrates how performance varies with the number of functions used to approximate curvature. As expected, increasing the number of functions enhances performance by enabling the model to capture multiple aspects of information propagation. However, this improvement is only observed up to a certain threshold, beyond which the additional complexity—introduced by an increased number of learnable parameters—begins to outweigh the benefits, ultimately leading to performance degradation. Notably, the model achieves good performance with a relatively small number of functions, which is advantageous from a computational efficiency perspective.
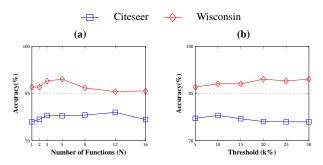

Figure 2: Model sensitivity to hyper-parameters

Figure 2 (b) illustrates how model performance varies with the threshold. For Citeseer, optimal performance is achieved at a lower

threshold (5–10%), whereas Wisconsin benefits from a higher threshold. This difference stems from dataset characteristics: Citeseer, being homophilic, has adjacent vertices with the same labels, allowing deeper aggregation to enhance information mixing. In contrast, Wisconsin, a heterophilic dataset, benefits from shallow depth aggregation to prevent mixing information from differently labeled neighboring vertices.
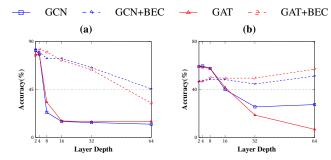

Figure 3: Oversmoothing comparison: (a) Cora; (b) Texas.

7.3.2 Oversmoothing Analysis. We analyze the impact of our adaptive layer depth approach on oversmoothing in the vertex classification task by progressively increasing the layer depth up to 64 while measuring classification accuracy. Notably, we set $k = 1$ to allow the aggregation stopping mechanism to remain effective even at large depths. As shown in Figure 3, our method exhibits substantial robustness against oversmoothing, achieving a significant performance margin over baseline models across both datasets.

Methods	Vertex Classification		Vertex Regression	
	Citeseer	Cornell	Network Science	
			IC	LT
GCN	76.68 ±1.64	57.03 ±4.67	0.270 ±0.019	0.190 ±0.012
GCN+ORC	78.49 ±0.35	57.83 ±4.22	0.251 ±0.006	0.170 ±0.023
GCN+FRC	78.51 ±0.47	58.92 ±4.80	0.253 ±0.005	0.178 ±0.024
GCN+JLC	77.03 ±0.56	58.64 ±3.20	0.239 ±0.010	0.155 ±0.011
GCN+ERC	78.70 ±0.66	59.72 ±2.54	0.249 ±0.007	0.155 ±0.029
GCN+BC	78.10 ±0.26	45.68 ±3.51	0.248 ±0.005	0.147 ±0.013
GCN+Degree	77.87 ±0.45	44.59 ±2.76	0.243 ±0.006	0.159 ±0.019
GCN+BEC	**80.43 ±0.88**	**61.62 ±4.64**	**0.233 ±0.010**	**0.123 ±0.026**

Table 4: Ablation study on curvature notions. For vertex classification, accuracy ± std(%) is reported; for vertex regression, MAE ± std is used. The best results are highlighted in bold.

7.3.3 Comparison with Existing Curvature Notions. In this experiment, we integrate existing curvature notions by replacing our learnable curvature with these predefined curvatures in our adaptive layer depth approach and evaluate their performance. We consider four curvatures from GNN literature—Ollivier-Ricci (**ORC**) [54, 58], Forman Ricci (**FRC**) [16], Jost and Liu (**JLC**) [19], and Effective Resistance (**ERC**) [10]—along with two structural descriptors, betweenness centrality (**BC**) and degree. These curvatures are

precomputed before training and interpreted consistently: higher values indicate denser local structures, while lower values correspond to sparser regions. Note that for any edge-based curvature, vertex curvature is computed as the average of adjacent edge curvatures. The results are summarized in Table 4.

Our approach, which learns curvature in a data-driven manner, significantly outperforms all precomputed curvatures in both downstream tasks by capturing task-specific information. Additionally, we compare our method with existing curvature-based rewiring approaches and demonstrate a substantial performance improvement (please see the Appendix).

7.3.4 Runtime and Parameter Complexity analysis. We evaluate scalability on large datasets, fixing the model depth at 4 and the hidden layer size at 256 for all methods. For our approach, we consider N in the set $\{1, 3, 5\}$. We adopt GCN and GIN as the base models for ogbn-arxiv and ogbg-moltox21, respectively. Each model is assessed on parameter count, average runtime (over 100 iterations), and accuracy/RoC, as shown in Table 5.

Methods	ogbn-arxiv			ogbg-moltox21		
	#Param (k)	Time (s)	Acc. (%)	#Param (k)	Time (s)	Roc. (%)
Baseline	176	0.30	70.87	1118	4.83	74.84
+BEC (N=1)	179	0.39	71.14	1123	6.90	75.22
+BEC (N=3)	184	0.52	71.76	1134	8.98	75.53
+BEC (N=5)	189	0.65	71.82	1144	10.39	75.76

Table 5: Runtime and parameter complexity comparison.

As N increases, parameter count and runtime grow moderately, ensuring computational feasibility. Our approach keeps parameter growth minimal by using MLPs to estimate curvature instead of expanding the GNN's parameter space. This design choice ensures that parameter learning of GNN remains efficient. With the same model capacity (i.e., fixed hidden layer size), our approach enhances representation power and outperforms the baseline, offering a meaningful improvement without excessive computational cost.

In the appendix, we present two additional experiments: (1) a visualization analysis, including 2D embedding visualization and curvature visualization of our approach, and (2) an evaluation of different threshold selection strategies, highlighting their influence on the depth-adaptive layer mechanism.

8 Conclusion, and Future Work

In this work, we propose a novel mechanism for GNNs that learns vertex-wise curvature in a data-driven manner. Our framework builds on Bakry-Émery curvature, which theoretically captures geometric intricacies and information diffusion in graphs from a functional perspective.By establishing a theoretical link between feature distinction in GNNs and vertex curvature, we introduce an adaptive layer depth mechanism that leverages vertex curvature to enhance the representation capacity of GNNs. Integrated with existing GNN architectures, the proposed approach consistently improves performance across diverse downstream tasks.

A limitation of our current approach is that the threshold controlling the adaptive layer depth mechanism is set as a hyperparameter. In future work, we aim to develop a method for learning this threshold directly from the graph structure and its underlying data distribution in an end-to-end manner.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. 2019. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *international conference on machine learning*. PMLR, 21–29.

[2] Luigi Ambrosio, Nicola Gigli, Giuseppe Savaré, et al. 2015. Bakry–Émery curvature-dimension condition and Riemannian Ricci curvature bounds. *ANNALS OF PROBABILITY* 43, 1 (2015), 339–404.

[3] Ismaël Bailleul and Frederic Bernicot. 2016. Heat semigroup and singular PDEs. *Journal of Functional Analysis* 270, 9 (2016), 3344–3452.

[4] Dominique Bakry and Michel Émery. 2006. Diffusions hypercontractives. In *Séminaire de Probabilités XIX 1983/84: Proceedings*. Springer, 177–206.

[5] Salomon Bochner. 1946. Vector fields and Ricci curvature. (1946).

[6] Shaked Brody, Uri Alon, and Eran Yahav. [n. d.]. How Attentive are Graph Attention Networks?. In *International Conference on Learning Representations*.

[7] Jialong Chen, Bowen Deng, Zhen WANG, Chuan Chen, and Zibin Zheng. 2025. Graph Neural Ricci Flow: Evolving Feature from a Curvature Perspective. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=7b2JrzdLhA

[8] Zhengdao Chen, Joan Bruna, and Lisha Li. 2019. Supervised community detection with line graph neural networks. In *7th International Conference on Learning Representations, ICLR 2019*.

[9] David Cushing, Shiping Liu, and Norbert Peyerimhoff. 2020. Bakry–Émery curvature functions on graphs. *Canadian Journal of Mathematics* 72, 1 (2020), 89–143.

[10] Karel Devriendt and Renaud Lambiotte. 2022. Discrete curvature on graphs from the effective resistance. *Journal of Physics: Complexity* 3, 2 (2022), 025008.

[11] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2023. Benchmarking Graph Neural Networks. *Journal of Machine Learning Research* 24, 43 (2023), 1–48.

[12] Federico Errica, Henrik Christiansen, Viktor Zaverkin, Takashi Maruyama, Mathias Niepert, and Francesco Alesiani. 2023. Adaptive Message Passing: A General Framework to Mitigate Oversmoothing, Oversquashing, and Underreaching. *arXiv preprint arXiv:2312.16560* (2023).

[13] Federico Errica, Marco Podda, Davide Bacciu, Alessio Micheli, et al. 2020. A Fair Comparison of Graph Neural Networks for Graph Classification. In *Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020)*.

[14] Lawrence C Evans. 2022. *Partial differential equations*. Vol. 19. American Mathematical Society.

[15] Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. 2022. How powerful are k-hop message passing graph neural networks. *Advances in Neural Information Processing Systems* 35 (2022), 4776–4790.

[16] Lukas Fesser and Melanie Weber. 2024. Mitigating over-smoothing and oversquashing using augmentations of Forman-Ricci curvature. In *Learning on Graphs Conference*. PMLR, 19–1.

[17] Ben Finkelshtein, Xingyue Huang, Michael M Bronstein, and Ismail Ilkan Ceylan. [n. d.]. Cooperative Graph Neural Networks. In *Forty-first International Conference on Machine Learning*.

[18] Robin Forman et al. 1999. Combinatorial differential topology and geometry. *New perspectives in geometric combinatorics* 38 (1999), 177–206.

[19] Jhony H Giraldo, Konstantinos Skianis, Thierry Bouwmans, and Fragkiskos D Malliaros. 2023. On the trade-off between over-smoothing and over-squashing in deep graph neural networks. In *Proceedings of the 32nd ACM international conference on information and knowledge management*. 566–576.

[20] Jacob Goldenberg, Barak Libai, and Eitan Muller. 2001. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters* 12 (2001), 211–223.

[21] Mark Granovetter. 1978. Threshold models of collective behavior. *American journal of sociology* 83, 6 (1978), 1420–1443.

[22] Mingjian Guang, Chungang Yan, Yuhua Xu, Junli Wang, and Changjun Jiang. 2024. Graph Convolutional Networks With Adaptive Neighborhood Awareness. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[24] Haoyu Han, Xiaorui Liu, Haitao Mao, MohamadAli Torkamani, Feng Shi, Victor Lee, and Jiliang Tang. 2023. Alternately optimized graph neural networks. In *International Conference on Machine Learning*. PMLR, 12411–12429.

[25] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[26] Kedar Karhadkar, Pradeep Kr Banerjee, and Guido Montufar. 2023. FoSR: First-order spectral rewiring for addressing oversquashing in GNNs. In *The Eleventh International Conference on Learning Representations*.

[27] DP Kingma. 2015. Adam: a method for stochastic optimization. In *International Conference on Learning Representations*.

[28] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations* (2017).

[29] John M. Lee. 2013. *Introduction to Smooth Manifolds* (2nd ed.). Graduate Texts in Mathematics, Vol. 218. Springer.

[30] Soo Yong Lee, Fanchen Bu, Jaemin Yoo, and Kijung Shin. 2023. Towards deep attention in graph neural networks: Problems and remedies. In *International Conference on Machine Learning*. PMLR, 18774–18795.

[31] Haifeng Li, Jun Cao, Jiawei Zhu, Yu Liu, Qing Zhu, and Guohua Wu. 2022. Curvature graph neural network. *Information Sciences* 592 (2022), 50–66.

[32] Yong Lin, Linyuan Lu, and Shing-Tung Yau. 2011. Ricci curvature of graphs. *Tohoku Mathematical Journal, Second Series* 63, 4 (2011), 605–627.

[33] Yong Lin and Shing-Tung Yau. 2010. Ricci curvature and eigenvalue estimate on locally finite graphs. *Mathematical research letters* 17, 2 (2010), 343–356.

[34] Chen Ling, Junji Jiang, Junxiang Wang, My T Thai, Renhao Xue, James Song, Meikang Qiu, and Liang Zhao. 2023. Deep graph representation learning and optimization for influence maximization. In *International Conference on Machine Learning*.

[35] Meng Liu, Haiyang Yu, and Shuiwang Ji. 2022. Empowering GNNs via Edge-Aware Weisfeiler-Leman Algorithm. *Transactions on Machine Learning Research* (2022).

[36] Yang Liu, Chuan Zhou, Shirui Pan, Jia Wu, Zhao Li, Hongyang Chen, and Peng Zhang. 2023. Curvdrop: A ricci curvature based approach to prevent graph neural networks from over-smoothing and over-squashing. In *Proceedings of the ACM Web Conference 2023*. 221–230.

[37] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3 (2000).

[38] Madhumita Mondal, Areejit Samal, Florentin Münch, and Jürgen Jost. 2024. Bakry–Émery–Ricci curvature: an alternative network geometry measure in the expanding toolbox of graph Ricci curvatures. *Journal of Complex Networks* 12, 3 (2024), cnae019.

[39] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. 2020. Tudataset: A collection of benchmark datasets for learning with graphs. *ICML workshop 'Graph Representation Learning and Beyond'* (2020).

[40] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and U Edu. 2012. Query-driven active surveying for collective classification. In *10th international workshop on mining and learning with graphs*, Vol. 8. 1.

[41] Khang Nguyen, Nong Minh Hieu, Vinh Duc Nguyen, Nhat Ho, Stanley Osher, and Tan Minh Nguyen. 2023. Revisiting over-smoothing and over-squashing using ollivier-ricci curvature. In *International Conference on Machine Learning*. PMLR, 25956–25979.

[42] Chien-Chun Ni, Yu-Yao Lin, Jie Gao, Xianfeng David Gu, and Emil Saucan. 2015. Ricci curvature of the internet topology. In *2015 IEEE conference on computer communications (INFOCOM)*. IEEE, 2758–2766.

[43] Yann Ollivier. 2007. Ricci curvature of metric spaces. *Comptes Rendus Mathematique* 345, 11 (2007), 643–646.

[44] Yann Ollivier. 2009. Ricci curvature of Markov chains on metric spaces. *Journal of Functional Analysis* 256, 3 (2009), 810–864.

[45] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations*.

[46] Maryam Pouryahya, Rena Elkin, Romeil Sandhu, Sarah Tannenbaum, Tryphon Georgiou, and Allen Tannenbaum. 2016. Bakry-Émery Ricci curvature on weighted graphs with applications to biological networks. In *Int. Symp. on Math. Theory of Net. and Sys*, Vol. 22. 52.

[47] Emanuele Rossi, Bertrand Charpentier, Francesco Di Giovanni, Fabrizio Frasca, Stephan Günnemann, and Michael M Bronstein. 2024. Edge directionality improves learning on heterophilic graphs. In *Learning on Graphs Conference*. PMLR, 25–1.

[48] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In *AAAI conference on artificial intelligence*, Vol. 29.

[49] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale attributed node embedding. *Journal of Complex Networks* 9, 2 (2021), cnab014.

[50] Avishkar Saha, Oscar Mendez, Chris Russell, and Richard Bowden. 2023. Learning adaptive neighborhoods for graph neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 22541–22550.

[51] Areejit Samal, RP Sreejith, Jiao Gu, Shiping Liu, Emil Saucan, and Jürgen Jost. 2018. Comparative analysis of two discretizations of Ricci curvature for complex networks. *Scientific reports* 8, 1 (2018), 8650.

[52] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.

[53] Stavros Souravlas, Sofia Anastasiadou, and Stefanos Katsavounis. 2021. A survey on the recent advances of deep community detection. *Applied Sciences* 11, 16 (2021), 7179.

[54] Joshua Southern, Jeremy Wayland, Michael M Bronstein, and Bastian Rieck. 2023. On the Expressive Power of Ollivier-Ricci Curvature on Graphs. (2023).

[55] Henan Sun, Xunkai Li, Zhengyu Wu, Daohan Su, Rong-Hua Li, and Guoren Wang. 2024. Breaking the entanglement of homophily and heterophily in semi-supervised node classification. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2379–2392.

[56] Li Sun, Zhenhao Huang, Hua Wu, Junda Ye, Hao Peng, Zhengtao Yu, and S Yu Philip. 2023. DeepRicci: Self-supervised Graph Structure-Feature Co-Refinement for Alleviating Over-squashing. In *2023 IEEE International Conference on Data Mining (ICDM)*. IEEE, 558–567.

[57] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 807–816.

[58] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. [n. d.]. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*.

[59] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[60] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[61] Frederic Weber and Rico Zacher. 2021. The entropy method under curvature-dimension conditions in the spirit of Bakry-Émery in the discrete setting of Markov chains. *Journal of Functional Analysis* 281, 5 (2021), 109061.

[62] Guofang Wei and Will Wylie. 2009. Comparison geometry for the Bakry-Emery Ricci tensor. *Journal of differential geometry* 83, 2 (2009), 337–405.

[63] Asiri Wijesinghe and Qing Wang. 2022. A new perspective on" how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*.

[64] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. PMLR, 6861–6871.

[65] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.

[66] Wenwen Xia, Yuchen Li, Jun Wu, and Shenghong Li. 2021. Deepis: Susceptibility estimation on social networks. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 761–769.

[67] Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. 2022. Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications* 33, 1 (2022), 4.

[68] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*.

[69] Ze Ye, Kin Sum Liu, Tengfei Ma, Jie Gao, and Chao Chen. 2019. Curvature graph network. In *International conference on learning representations*.

[70] Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 10737–10745.

[71] Taraneh Younesian, Thiviyan Thanapalasingam, Emile van Krieken, Daniel Daza, and Peter Bloem. [n. d.]. GRAPES: Learning to Sample Graphs for Scalable Graph Neural Networks. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*.

[72] Bingxu Zhang, Changjun Fan, Shixuan Liu, Kuihua Huang, Xiang Zhao, Jincai Huang, and Zhong Liu. 2024. The expressive power of graph neural networks: A survey. *IEEE Transactions on Knowledge and Data Engineering* (2024).

[73] Muhan Zhang. 2022. Graph neural networks: link prediction. *Graph Neural Networks: Foundations, Frontiers, and Applications* (2022), 195–223.

[74] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in neural information processing systems* 31 (2018).

[75] Weichen Zhao, Tiande Guo, Xiaoxi Yu, and Congying Han. 2023. A learnable sampling method for scalable graph neural networks. *Neural Networks* 162 (2023), 412–424.

[76] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in neural information processing systems* 33 (2020), 7793–7804.

Appendix

A Proofs

THEOREM 4.1 (MIXING TIME BOUND). *Let $G = (V, E, w)$ be an undirected, weighted graph with bounded degree and Laplacian L. If the local curvature-dimension inequality holds for every $f \in \ell^2(V)$ at vertex x, then for any $\epsilon \in (0, 1)$, the local mixing time satisfies*

$$\tau_x(\epsilon) \leq \frac{\log(1/\epsilon)}{\kappa(x)}.$$

PROOF. First, we explicitly define the operators [33]:

$$\Gamma(f, g)(x) = \frac{1}{2} \left(L(fg) - fLg - gLf \right)(x)$$

$$\Gamma_2(f, f)(x) = \frac{1}{2} \left(L\Gamma(f, f)(x) - \Gamma(f, Lf)(x) \right)$$

Under the heat semigroup evolution $f_t = e^{-tL} f_0$ [3], for any initial function $f_0 \in \ell^2(V)$, we have:

$$\frac{d}{dt} |\nabla f_t|^2(x) = -2\Gamma(f_t, Lf_t)(x)$$

Here, we differentiate the squared gradient with respect to time. The operator $\Gamma(f_t, Lf_t)(x)$ represents the interaction between the graph Laplacian and the gradient of the function at each point in time. Next, we apply the local Γ_2-criterion, which gives the lower bound:

$$\Gamma_2(f_t, f_t)(x) \geq \kappa(x) |\nabla f_t|^2(x)$$

This criterion tells us that the curvature $\kappa(x)$ at each vertex provides a lower bound for the second-order derivative of the function f_t in terms of the gradient squared. Using this inequality, we obtain:

$$\frac{d}{dt} |\nabla f_t|^2(x) \leq -2\kappa(x) |\nabla f_t|^2(x)$$

The negative sign indicates that the gradient squared decreases over time due to the decay effect imposed by the curvature $\kappa(x)$. This step shows how the curvature influences the rate of change of the gradient over time. Now, we apply Grönwall's inequality [14], a standard tool for solving differential inequalities:

$$|\nabla f_t|^2(x) \leq e^{-2\kappa(x)t} |\nabla f_0|^2(x)$$

Grönwall's inequality gives an upper bound for the evolution of $|\nabla f_t|^2(x)$ based on the initial condition $|\nabla f_0|^2(x)$. The exponential decay reflects the influence of the curvature $\kappa(x)$ on the gradient. To achieve $|\nabla f_t|^2(x) \leq \epsilon |\nabla f_0|^2(x)$ for any $\epsilon > 0$, we solve for t to get:

$$t \geq \frac{\log(1/\epsilon)}{2\kappa(x)}$$

This bound provides the required time to ensure that the gradient squared at each vertex decays to the desired level. The time t depends on the initial condition and the local curvature at each vertex. Finally, we conclude that the local mixing time $\tau_x(\epsilon)$, which represents the time required for the gradient to decay by a factor of ϵ, satisfies the inequality:

$$\tau_x(\epsilon) \leq \frac{\log(1/\epsilon)}{\kappa(x)}$$

This inequality holds for any initial condition $f_0 \in \ell^2(V)$, where $\ell^2(V)$ is the space of square-summable functions, meaning that

$\sum_{x \in V} |f_0(x)|^2 < \infty$. This ensures that the function f_0 has finite energy, and the solutions to $f_t = e^{-tL} f_0$ are well-defined for all $t \geq 0$.

$\square$

THEOREM 6.1 (FEATURE DECAY BOUND). *Let $G = (V, E, w)$ be an undirected, weighted graph with bounded degree. Consider a GNN with l layers, where each layer approximates the heat flow for a time step Δt. For a vertex $x \in V$ with Bakry-Émery curvature $\kappa(x)$, we define the feature distinctiveness $D(x, l)$ after l layers as*

$$D(x, l) = \frac{|\nabla f_l|^2(x)}{|\nabla f_0|^2(x)},$$

where f_l represents the vertex features at layer l. Then, the following holds:

$$D(x, l) \leq e^{-\kappa(x)l\Delta t}.$$

For any $\epsilon > 0$, to maintain $D(x, l) \geq \epsilon$, it suffices that the number of layers satisfy:

$$l \leq \frac{\log(1/\epsilon)}{\kappa(x)\Delta t}.$$

PROOF. From the definition of $\tau_x(\epsilon)$ in Theorem 4.1, we know that:

$$|\nabla f_t|^2(x) \leq \epsilon |\nabla f_0|^2(x) \quad \text{for all} \quad t \geq \tau_x(\epsilon).$$

Using the upper bound for $\tau_x(\epsilon)$, we have:

$$\tau_x(\epsilon) \leq \frac{\log(1/\epsilon)}{\kappa(x)}.$$

Thus, for all $t \geq \frac{\log(1/\epsilon)}{\kappa(x)}$, it follows that:

$$|\nabla f_t|^2(x) \leq \epsilon |\nabla f_0|^2(x).$$

Now, to express this in the desired form, observe that the exponential decay bound for $|\nabla f_t|^2(x)$ is given by:

$$|\nabla f_t|^2(x) \leq |\nabla f_0|^2(x) \exp(-\kappa(x)t).$$

Each layer in the GNN approximates the heat flow for a time step Δt, so after l layers, the time is $t = l\Delta t$. Substituting this into the definition of $D(x, l)$, we get:

$$D(x, l) = \frac{|\nabla f_l|^2(x)}{|\nabla f_0|^2(x)} \leq \exp(-\kappa(x)l\Delta t).$$

For the second part, to maintain $D(x, l) \geq \epsilon$, we require:

$$\exp(-\kappa(x)l\Delta t) \geq \epsilon.$$

Taking the logarithm of both sides:

$$-\kappa(x)l\Delta t \geq \log(\epsilon),$$

which simplifies to:

$$l \leq \frac{\log(1/\epsilon)}{\kappa(x)\Delta t}.$$

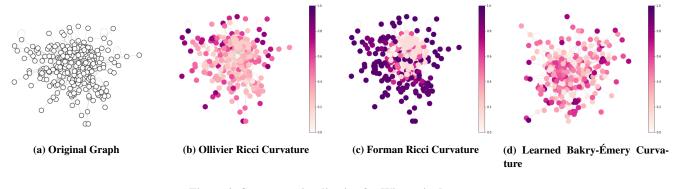Thus, the number of layers must satisfy the given bound to ensure that $D(x, l) \geq \epsilon$. $\square$

(a) Original Graph   (b) Ollivier Ricci Curvature   (c) Forman Ricci Curvature   (d) Learned Bakry-Émery Curvature

Figure 4: Curvature visualization for Wisconsin dataset

(a) MixHop   (b) MixHop+ORC   (c) MixHop+FRC   (d) MixHop+BEC

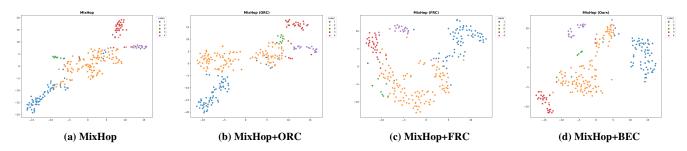**Figure 5: 2D Vertex embedding visualization for Wisconsin dataset**

B Supplementary Experimental Details

B.1 Downstream Tasks

We evaluate our model in the following downstream tasks.

Vertex classification: The goal of this task is to predict the label of individual vertices in a graph, utilizing both the vertex features and the structural information of the graph.

Vertex regression: Vertex regression aims to predict continuous values for individual vertices in a graph. In our work, we focus on influence estimation [34, 66], a fundamental and challenging problem in network science. Specifically, we consider a diffusion model with an initial set of activated vertices and seek to predict the activation probability of each vertex once the diffusion process has converged. This task involves GNN leveraging the graph's structure and adapting the dynamics of the diffusion process to accurately estimate the likelihood of vertex activation.

Graph classification: Graph classification involves predicting a single label for an entire graph based on its structure and vertex features.

Graph regression: Graph regression involves predicting continuous values for an entire graph, with the goal of estimating a scalar value that characterizes the graph's overall properties.

B.2 Baselines

We employ the following baselines for our evaluations.

(1) Vertex classification and regression

- **GCN** [28] utilizes spectral graph convolution to aggregate neighborhood vertex features.

- **GraphSAGE** [23] incorporates neighborhood sampling and supports various aggregation functions (e.g., mean, LSTM, or sum) to enhance scalability.

- **GAT** [60] employs attention mechanisms to assign adaptive weights to neighboring vertices, enabling more effective message aggregation.

- **SGC** [64] simplifies GCN by eliminating nonlinearities and collapsing weight matrices, resulting in a more efficient linear model.

- **MixHop** [1] aggregates information from multiple-hop neighborhoods, capturing higher-order interactions within the graph.

- **DIR-GNN** [47] is a heterophilic-specific GNN that improves the message passing by incorporating directional information.

(2) Graph classification and regression

- **GIN** [68] employs MLPs and sum aggregations to distinguish different graph structures, with its distinguishing power upper-bounded by the 1-WL algorithm.

- **GCN** [28] is a GNN based on spectral graph convolution, and its graph distinguish power is upper-bounded by 1-WL algorithm.

- **ID-GNN** [70] enhances graph distinguishability by incorporating vertex identity as a coloring mechanism, achieving expressiveness beyond the 1-WL algorithm.

- **GraphSNN** [63] injects overlapping subgraph-based structural information into message passing, enhancing its expressiveness beyond the 1-WL algorithm.

- **NC-GNN** [35] incorporates edges between neighboring vertices, in addition to neighbors, in its message-passing step, resulting in higher expressive power compared to the 1-WL algorithm.

B.3 Model Hyper-Parameters

In our experiments, we search for hyper-parameters within the following ranges: the number of layers is selected from $\{2, 3, 4, 5\}$, k from $\{5, 10, 15, 20, 30, 40\}$, N from $\{1, 2, 3, 4, 5\}$, dropout from $\{0.0, 0.1, 0.2, 0.5, 0.6, 0.9\}$, learning rate from $\{0.005, 0.009, 0.01, 0.1\}$, batch size from $\{32, 64\}$, hidden units from $\{32, 64, 128, 256, 300\}$, weight decay from $\{5e-4, 9e-3, 1e-2, 1e-1\}$, and the number of epochs from $\{100, 200, 500\}$. The Adam optimizer [27] is used for optimization. Further, we employ $\lambda = 1$ for all the experiments. For the ZINC and ogbg-moltox21 datasets, the learning rate decays by a factor of 0.5 after every 10 epochs.

B.4 Benchmark Dataset Statistics

Tables 6 and 7 present an overview of the statistics for the datasets used in our experiments. Specifically, Table 6 covers statistics for vertex classification and regression, while Table 7 provides dataset statistics related to graph classification and regression tasks.

B.5 Computational Resources

All the experiments were carried out on a Linux server with an Intel Xeon W-2175 2.50GHz processor (28 cores), an NVIDIA RTX A6000 GPU, and 512GB of RAM.

B.6 Additional Experiments

B.6.1 Visualization Analysis. We present a visualization analysis of vertex curvature for differnt curvature notions in Figure 4 using the Wisconsin dataset. In our approach, curvature is learned with MixHop as the base model. For two Ricci curvature methods, vertex curvature is computed as the average of adjacent edge curvatures. All the vertex curvature values are normalized between 0 and 1 for visualization purposes. The comparison reveals that our method captures heterogeneous curvature patterns with greater granularity compared to other approaches. This level of detail is well justified, as Wisconsin is a heterophilic dataset.

Figure 5 provides vertex embedding visualization using t-SNE [59]. MixHop is employed as the base model, and we evaluate its vertex classification performance on the Wisconsin dataset. The colors represent different class labels in the dataset. The comparison of embeddings shows significant improvements in our approach. The most notable enhancement is the formation of well-separated clusters. Our method improves cluster boundary definition, reducing overlap between class labels. Overall, our method captures both fine-grained relationships and broader community structures in embedding, leading to better classification. This is well supported by the curvature visualization, as our approach effectively captures heterogeneous curvature patterns that other methods fail to detect.

B.6.2 Comparison with Curvature-based Rewiring Approaches. In this experiment, we compare our approach with existing curvature-driven rewiring methods for vertex classification. We employ the experimental setup followed by Fesser and Weber [16] and evaluate four rewiring techniques: AFR [16], BORF [41], SDRF [58], and FOSR [26].

Method	Cora	Citeseer	Texas	Cornell	Wisconsin
GCN	86.6 ±0.8	71.7 ±0.7	44.1 ±0.5	46.8 ±3.0	44.6 ±2.9
GCN+AFR	**88.1 ±0.5**	74.4 ±1.0	51.4 ±0.5	49.7 ±3.4	52.2 ±2.4
GCN+BORF	87.9 ±0.7	73.4 ±0.6	48.9 ±0.5	48.1 ±2.9	46.5 ±2.6
GCN+SDRF	86.4 ±2.1	72.6 ±2.2	43.6 ±1.2	43.1 ±1.2	47.1 ±1.0
GCN+FOSR	86.9 ±2.0	73.5 ±2.0	46.9 ±1.2	43.9 ±1.1	48.5 ±1.0
GCN+BEC	86.9 ±0.3	**77.3 ±0.4**	**64.4 ±2.5**	**52.9 ±3.1**	**66.9 ±2.4**

Table 8: Vertex classification accuracy ± std (%) comparison with curvature-based rewiring approaches. Baseline results are taken from Fesser and Weber [16]. The best results are highlighted in bold.

As shown in Table 8, our method significantly outperforms these existing approaches, highlighting that incorporating curvature directly into the learning pipeline provides a notable performance boost, compared to integrating it solely into the rewiring process.

B.6.3 Comparison with different threshold selection mechanisms. One of the key design decisions in our adaptive layer mechanism is determining the threshold for each layer. To gain better empirical insights into this decision, we conduct an experiment for vertex classification using GCN as the base model, with a depth of 20 layers. In this experiment, the threshold k in each layer is determined by selecting from four common distributions: **Fixed**, where k remains constant across all layers; **Power-law**, where k is sampled from a power-law distribution; **Normal (Gaussian)**, where k is sampled from a normal (Gaussian) distribution; and **Linear (increasing)**, where k increases progressively across layers. The results of this experiment are presented in Figure 6.
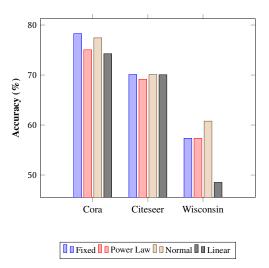

Figure 6: Comparison of different threshold selection mechanisms

Based on the results, the fixed and normal distributions exhibit superior performance compared to the other approaches. However, it is well-established that GNNs generally perform better with shallow layers. Since it is challenging to effectively sample from a normal distribution with a limited number of samples, we opt for the Fixed threshold. This choice is not only more practical but also simple.

Dataset	Task Type	# Vertices	# Edges	# Classes
Jazz	Vertex Regression	198	2,742	1
Network Science	Vertex Regression	1,565	13,532	1
Cora-ML	Vertex Regression	2,810	7,981	1
Power Grid	Vertex Regression	4,941	6,594	1
Cora	Vertex Classification	2,708	5,429	7
Citeseer	Vertex Classification	3,327	4,732	6
Pubmed	Vertex Classification	19,717	44,338	3
Actor	Vertex Classification	7,600	33,544	5
Squirrel	Vertex Classification	5,201	217,073	5
Wisconsin	Vertex Classification	251	499	5
Cornell	Vertex Classification	183	295	5
Texas	Vertex Classification	183	309	5
ogbn-arxiv	Vertex Classification	169,343	1,166,243	40

Table 6: Dataset statistics for vertex classification and regression.

Datasets	Task Type	# Graphs	Avg #Vertices	Avg #Edges	# Classes
MUTAG	Graph Classification	188	17.93	19.79	2
PTC_MR	Graph Classification	344	14.29	14.69	2
BZR	Graph Classification	405	35.75	38.36	2
COX2	Graph Classification	467	41.22	43.45	2
IMDB-BINARY	Graph Classification	1,000	19.77	96.53	2
PROTEINS	Graph Classification	1,113	39.06	72.82	2
ogbg-moltox21	Graph Classification	7,831	18.6	19.3	2
ZINC	Graph Regression	12,000	23.1	49.8	1

Table 7: Dataset statistics for graph classification and regression.