Do We Need Curved Spaces? A Critical Look at Hyperbolic Graph Learning in Graph Classification

Dionisia Naddeo[1], Tiago Azevedo[3], and Nicola Toschi[1,2]

[1] Department of Biomedicine and Prevention, University of Rome, Tor Vergata Italy
[2] A.A. Martinos Center for Biomedical Imaging, Harvard Medical School Boston USA
[3] Department of Computer Science and Technology, University of Cambridge

Abstract. Hyperbolic geometry has gained attention for its ability to naturally embed hierarchical and tree-like structures with low distortion, outperforming Euclidean spaces in various graph representation tasks. While previous work has demonstrated the advantage of graph neural networks embedding in hyperbolic space for link prediction and node classification, the benefits for graph classification, and especially the role of node features, remain less understood. These studies typically attribute the benefits of hyperbolic models to the hierarchical or tree-like nature of graph structures, often neglecting the important role that node features play in leveraging these geometric advantages. With this in mind, we design an experiment specifically aimed at evaluating the interplay between geometry and node features in graph classification, creating a dataset composed exclusively of tree-structured graphs. Each graph is generated by sampling the number of children per node at each level from a predefined range of branching factors, which varies across levels. The dataset defines two distinct classes based on these branching factor patterns. Node features are either *random*, structural embeddings obtained via *node2vec* (using BFS-biased walks), or layout-based embeddings derived from *kamada-kawai*, which provide a strong hierarchical prior. We evaluated and compared graph neural networks with node embeddings learned in different geometries — hyperbolic vs. Euclidean — in low-dimensional latent spaces. Results show that Euclidean models consistently outperform hyperbolic models across all feature types. This suggests that the advantage of hyperbolic geometry does not stem solely from alignment with global graph structure. These findings call for a critical reassessment of hyperbolic models in supervised tasks where preserving graph distances is not essential.

Keywords: Hyperbolic Graph Learning · Graph Neural Networks · Lorentz Model · Synthetic Dataset

1 Introduction

In network science, hyperbolic spaces have emerged as powerful tools for embedding hierarchical data. Such data often exhibit a tree-like structure, where entities are organized in levels, originating from general to specific categories, or from parent to child nodes. Trees naturally model these hierarchies, making them a canonical example of hierarchical structures.

Early studies have shown that hyperbolic geometry can outperform its Euclidean counterpart in capturing such structures within the Poincaré [14] and Lorentz [15] models, and the Lorentz model often achieves superior empirical performance. This advantage comes from a fundamental geometric property: the metric structures of trees and hyperbolic spaces are aligned [9]. In tree-like graphs [18], the number of nodes increases exponentially with distance from the root, and hyperbolic space mirrors this property, as both the circumference of a circle and the area of a disk grow exponentially with radius. Consequently, hyperbolic geometry offers a natural setting for embedding trees and hierarchical graphs, particularly in low-dimensional spaces: while a tree can be embedded with low distortion in a two-dimensional hyperbolic manifold, this is generally not possible in Euclidean space of the same dimension because the volume of a disk, i.e. the number of points reachable within a certain radius, grows polynomially, as r^d, where d is the dimension of the space. However, as the dimension increases, trees can be embedded with less distortion.

Building upon these insights, subsequent research has extended Graph Neural Networks (GNNs)—particularly graph convolutional architectures—into hyperbolic space, with the aim of modeling graphs that exhibit tree-like structures more effectively. Much of this research has focused on link prediction and node classification tasks [2, 12, 20], demonstrating that hyperbolic embeddings can preserve graph topology with low distortion. While most existing works attribute the advantage of hyperbolic space to the hierarchical nature of the graph structure, they often overlook the role of node features. Specifically, graph convolutions operate primarily on node features, effectively treating them as proxies for the nodes themselves. However, in a geometric context, it is important to recognize that the benefits of hyperbolic embeddings only hold when nodes are embedded explicitly with respect to their original graph distances. Therefore, the modeling of node features should also be investigated with precision [7].

This limitation is partially mitigated in link prediction, where the use of the Fermi-Dirac decoder [9] encourages node embeddings to reflect the geodesic structure of the graph. By assigning higher edge probabilities to node pairs that are closer in hyperbolic space, the decoder promotes meaningful separation of embeddings based on graph connectivity.

However, in graph classification tasks, node features are typically aggregated to form a graph embedding - either directly in hyperbolic space or after being mapped to Euclidean space, for example, by distances to learnable centroids [13, 16] - and then passed to fully connected layers for prediction. In addition, standard graph classification objectives, such as binary cross-entropy loss, optimize label accuracy but do not explicitly encourage the preservation of the graph's topological or hierarchical structure in node embeddings. It is therefore not evident why hyperbolic representations should offer an advantage for graph-level tasks where tree-like structure is not explicitly enforced during training unless node features themselves encode such hierarchical relationships. In light of this, our aim is to investigate how hyperbolic graph models behave when node features explicitly encode, or do not encode, the hierarchical structure of the graph.

In this work, we design an experiment to evaluate hyperbolic GNNs for graph classification by introducing a novel benchmark data set of synthetic tree-structured graphs. Previous research [7] proposed a tree-like dataset primarily intended for link prediction tasks. However, it is not directly applicable to graph classification, as it does not define any class labels. Existing graph classification benchmarks for hyperbolic networks [13] distinguish graphs generated by distinct algorithms such as Erdős-Rényi [4], Barabási-Albert [1], and Watts-Strogatz [19]. In contrast, we design a dataset that defines explicit class labels by generating tree-structured graphs from predefined branching factor patterns. This design introduces controlled hierarchical variations between classes, enabling a focused evaluation of graph classification methods in settings where hierarchical structure is both present and class-defining.

In order to examine the role of node features in hyperbolic graph neural networks, we construct three types of node features: *random* features, which do not encode any structural information, *node2vec* features, which capture the structural context of nodes by reflecting their roles and neighborhood patterns within the graph and *kamada-kawai* features. We then compare the performance of hyperbolic GNNs to Euclidean GNNs.

Our contributions are as follows:

- We propose a novel synthetic benchmark dataset of tree-structured graphs with explicit hierarchical disruptions to rigorously evaluate hyperbolic graph neural networks in graph classification tasks.
- We identify a critical gap in existing hyperbolic GNN research on the role of hierarchical information in node features in graph classification tasks and design node feature types that either ignore or emphasize local hierarchical structure.
- We systematically compare hyperbolic and Euclidean GNNs using low-dimensional embeddings (3D Euclidean vs. 2D hyperbolic via the hyperboloid model in $\mathbb{R}^3$), demonstrating that the hyperbolic model does not yield performance gains.

2 Hyperbolic Graph Neural Network

For a brief summary of Riemannian manifolds and hyperbolic geometry, we refer the reader to Appendix A and to the references therein.

To model node embeddings in hyperbolic space, we adopt HyboNet [3], a state-of-the-art hyperbolic graph neural network that operates entirely in the Lorentz model. Feature transformation and non-linear activation are combined in a Lorentz Linear Layer:

$$\mathbf{y} = \mathrm{HL}(\mathbf{x}) = \begin{bmatrix} \sqrt{\|\varphi(\mathbf{W}\mathbf{x}, \mathbf{v})\|^2 - \frac{1}{c}} \\ \frac{\varphi(\mathbf{W}\mathbf{x}, \mathbf{v})}{\|\varphi(\mathbf{W}\mathbf{x}, \mathbf{v})\|} \end{bmatrix}, \tag{1}$$

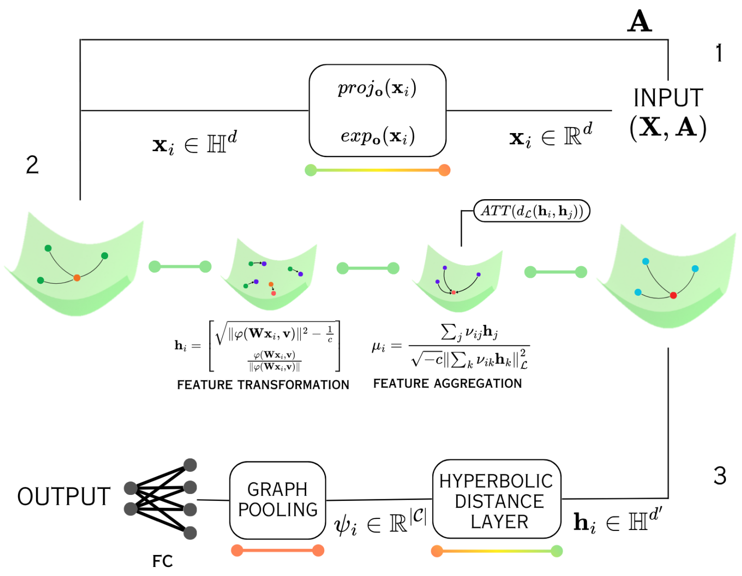
A
1
INPUT
$(\mathbf{X}, \mathbf{A})$
$\mathbf{x}_i \in \mathbb{R}^d$
$proj_{\mathbf{o}}(\mathbf{x}_i)$
$exp_{\mathbf{o}}(\mathbf{x}_i)$
$\mathbf{x}_i \in \mathbb{H}^d$
2
$ATT(d_{\mathcal{L}}(\mathbf{h}_i, \mathbf{h}_j))$
$\mathbf{h}_i = \begin{bmatrix} \sqrt{\|\varphi(\mathbf{W}\mathbf{x}_i, \mathbf{v})\|^2 - \frac{1}{c}} \\ \frac{\varphi(\mathbf{W}\mathbf{x}_i, \mathbf{v})}{\|\varphi(\mathbf{W}\mathbf{x}_i, \mathbf{v})\|} \end{bmatrix}$
FEATURE TRANSFORMATION
$\mu_i = \frac{\sum_j \nu_{ij}\mathbf{h}_j}{\sqrt{-c}\|\sum_k \nu_{ik}\mathbf{h}_k\|_{\mathcal{L}}^2}$
FEATURE AGGREGATION
3
$\mathbf{h}_i \in \mathbb{H}^{d'}$
HYPERBOLIC DISTANCE LAYER
$\psi_i \in \mathbb{R}^{|\mathcal{C}|}$
GRAPH POOLING
OUTPUT
FC

Fig. 1: Pipeline of the Hyperbolic Graph Neural Network (HGNN) for graph classification. Colored lines indicate the geometric space in which each operation takes place: orange for Euclidean space, green for hyperbolic space, and gradients for transitions between them. The pipeline proceeds as follows: (1) The input pair $(\mathbf{X}, \mathbf{A})$, consisting of node features and the adjacency matrix, is mapped to hyperbolic space via the exponential map from the tangent space at the origin. (2) A hyperbolic graph neural network processes the data, performing feature transformation and aggregation entirely in hyperbolic space. Attention weights are computed using Lorentzian distances between node embeddings. (3) Final node embeddings are passed through a hyperbolic distance layer and mapped back to Euclidean space. These embeddings are then pooled into a graph-level representation and passed through a fully connected layer for classification.

where the input $\mathbf{x} \in \mathbb{L}_c^n$ lies on the Lorentz manifold with curvature $c < 0$, and $\mathbf{v} \in \mathbb{R}^{n+1}$ and $\mathbf{W} \in \mathbb{R}^{m\times(n+1)}$ are learnable parameters. The function $\varphi(\cdot)$ represents a transformation that includes dropout, activation, bias, and normalization. For details, we refer the reader to the original publication [3].

Neighborhood aggregation is performed using the center of mass in Lorentz space, with attention computed via the Lorentzian distance $d_{\mathcal{L}}(\cdot, \cdot)$. Given node embeddings q_i (queries), k_j (keys), and v_j (values), the attention output is

$$\mu_i = \frac{\sum_j \nu_{ij} v_j}{\sqrt{-c}\,\|\sum_k \nu_{ik} v_k\|_{\mathcal{L}}^2}, \tag{2}$$

where the attention weights are

$$\nu_{ij} = \frac{\exp\left(-\frac{d_{\mathcal{L}}(q_i,k_j)}{\sqrt{n}}\right)}{\sum_k \exp\left(-\frac{d_{\mathcal{L}}(q_i,k_k)}{\sqrt{n}}\right)}. \tag{3}$$

The HyboNet encoder outputs node embeddings on the Lorentz manifold. For graph-level classification, we follow [13] and use a set of learnable centroids $\mathcal{C} = [\mathbf{c}_1, \ldots, \mathbf{c}_{|\mathcal{C}|}]$, where each centroid $\mathbf{c}_i \in \mathbb{L}_c^n$ lies in the Lorentz curvature manifold c. The centroids $\{\mathbf{c}_i\}$ are optimized jointly with the GNN parameters via backpropagation. For each centroid $\mathbf{c}_i$, we compute the distances to all node embeddings $\mathbf{h}_j$ as $\psi_{ij} = d(\mathbf{c}_i, \mathbf{h}_j)$, and average over all nodes to obtain

$$\psi_i = \frac{1}{N}\sum_{j=1}^{N} \psi_{ij}.$$

The final global graph embedding is the vector

$$\boldsymbol{\psi} = (\psi_1, \ldots, \psi_{|\mathcal{C}|}) \in \mathbb{R}^{|\mathcal{C}|},$$

that is used as a global representation of the graph and is fed into a fully connected layer for classification. A detailed schematic of the pipeline is shown in Fig.(1).

3 Dataset

We design a synthetic dataset for graph classification composed exclusively of tree-like structures, where the distinguishing factor between classes is explicitly rooted in hierarchical organization. Each graph belongs to one of two classes, and all graphs contain the same number of nodes. The two classes differ in their branching factor patterns across levels, a property we deliberately perturbed as it plays a central role in shaping the hierarchical organization of the network.

We create two versions of the dataset: *Easy* and *Hard*. In both cases, the number of nodes is fixed to 200, and the dataset is split into 600 training, 200 validation, and 200 test samples.

3.1 Graph Generation

Each tree is generated by specifying a list of branching factor ranges — one range per level. At level i, each parent node samples a random number of children from the interval $[r_i^{\min}, r_i^{\max}]$. The generation proceeds level by level until the desired number of nodes is reached. This process yields a flexible hierarchical structure, where the global shape of each tree depends on the branching ranges. After generation, we ensured that no two graphs were isomorphic using the method descibed in [17].

In the *Easy* setting, class A uses a fixed branching range of $(4, 6)$ at each level, while class B uses $(1, 3)$. This results in two structurally and visually distinct tree topologies: one with high branching and the other with low branching. As shown in Fig. 2 (left), the two classes are clearly separable. We quantified this structural difference using several graph-level metrics computed with the *NetworkX* library: tree depth, diameter, radius, average shortest path length, and degree entropy.

In the *Hard* setting, the class distinction becomes more subtle. Both class A and class B share a common base branching scheme defined by the following list of ranges:

$$[(4, 7),\ (4, 6),\ (3, 5),\ (3, 4),\ (2, 3),\ (1, 2)],$$

with the range $(1, 2)$ applied to all subsequent levels until the graph reaches the target number of nodes. These ranges were carefully chosen to ensure the number of nodes grows approximately exponentially from the root, while also maintaining high variability between samples within each class. In class B, we introduce local mutations at levels 2 and 3, where the branching factor for each parent node is changed to $(1, 2)$ with probability 0.5. This mutation mechanism injects localized structural variability while preserving the global hierarchical shape, making the classification task more challenging. Unlike the *Easy* setting, in the *Hard* case the structural graph metrics across classes are more aligned, though not entirely indistinguishable, as illustrated in Fig.2 (right). This increased similarity in global metrics requires models to rely on more subtle and non-trivial structural cues for successful classification. The dataset generation process offers high tunability with respect to task difficulty, allowing precise control over structural variation. In Appendix B, we demonstrate this flexibility by introducing an *Extra Hard* version of the task, where the graph metrics become virtually indistinguishable across classes.

Task	**Node Feature**	**Euclidean**	**Hyperbolic**
Easy	random	-0.0050 ± 0.0288	-0.0070 ± 0.0407
Easy	node2vec	0.6922 ± 0.0630	0.7244 ± 0.0616
Easy	kamada-kawai	0.9754 ± 0.0128	0.8627 ± 0.0264
Hard	random	-0.0014 ± 0.0352	-0.0012 ± 0.0492
Hard	node2vec	0.7723 ± 0.0251	0.8188 ± 0.0208
Hard	kamada-kawai	0.9656 ± 0.0108	0.8559 ± 0.0212

Table 1: Spearman correlation between graph-theoretic distances and pairwise node embedding distances, computed in Euclidean and hyperbolic space. Results are shown for two tasks (*Easy* and *Hard*) and two node feature types: *random* (Gaussian noise) and *node2vec* embeddings with $(p = 1, q = 4)$ to bias walks toward local neighborhoods.

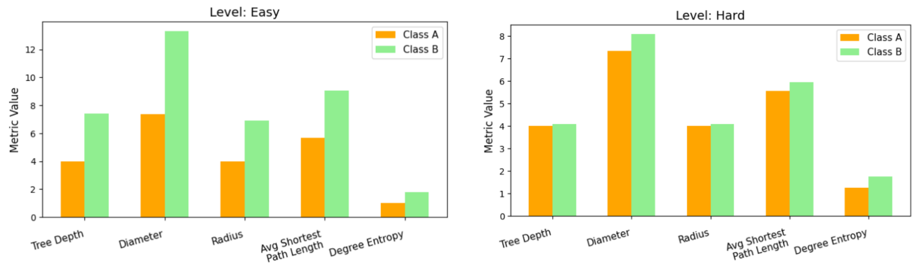

Fig. 2: Structural differences between the *Easy* and *Hard* synthetic datasets. The histograms show the average value of the each metric for each class. **Left:** In the *Easy* setting, class A and class B trees have clearly distinct branching patterns resulting in separable topologies. **Right:** In the *Hard* setting, both classes share a similar global structure.

Task	**Node Feature**	**Euclidean**	**Hyperbolic**
Easy	node2vec	0.5444 ± 0.1558	0.5442 ± 0.1559
Hard	node2vec	0.7150 ± 0.0783	0.7142 ± 0.0780
Easy	kamada-kawai	0.6788 ± 0.2927	0.6788 ± 0.2927
Hard	kamada-kawai	0.8550 ± 0.1574	0.8550 ± 0.1574

Table 2: Spearman correlation between node hierarchy levels (depth from the root) and node distances from the origin in the embedding space, for both Euclidean and hyperbolic geometries, using *node2vec* and *kamada-kawai* features. Results are shown for the *Easy* and *Hard* tasks. Higher values indicate stronger alignment between node depth and embedding position, reflecting better encoding of hierarchical structure.

3.2 Node Feature Construction

We construct two types of node features with varying levels of structural information to evaluate how effectively hyperbolic space can capture these signals. We assume that structural information implicitly captures the hierarchical relationships of nodes within the graph.

We considered the following types of node features, each of dimension 200:

– **Random features:** Each node is assigned a feature vector sampled from a multivariate normal distribution with zero mean and variance ($\sigma = 0.5$), resulting in a narrow distribution that contains no structural information.
– **Node2Vec features:** We extract node embeddings using the *node2vec* algorithm [5], which can capture both local and global network topology through biased random walks. The algorithm learns a mapping of nodes to a low-dimensional space by maximizing the likelihood of preserving network neighborhoods. We set the walk length to 30 and the number of walks per node

to 20. We set the return parameter p to 1.0 and the in-out parameter q to 4.0, biasing the random walks toward a BFS-like behavior. This local bias encourages the embeddings to better capture nodes' hierarchical roles within their neighborhoods, which may be especially beneficial for the *Hard* task, where subtle, localized structural mutations distinguish classes.

– **Kamada-Kawai features:** We compute node embeddings using the *Kamada-Kawai* layout algorithm, which positions nodes in an n-dimensional space by minimizing an energy function based on graph-theoretic distances (i.e., shortest paths). This method models the graph as a spring system, where edges act like springs and ideal distances between nodes are proportional to their shortest path lengths. The resulting coordinates reflect the global structure of the graph, allowing embeddings to capture topological roles and relative positions within the hierarchy. We use the `NetworkX` implementation with `dim=200` and store the resulting 200-dimensional coordinates as node features. The coordinates are then normalized across dimensions (i.e., feature-wise) within each graph.

To initialize the node features in hyperbolic space, we apply the exponential map from the tangent space at the origin to the hyperboloid model (Eq.(5) in Appendix A). This procedure allows us to embed Euclidean feature vectors into hyperbolic space.

To assess the extent to which node features capture hierarchical information, we compute the Spearman correlation between pairwise graph-theoretic distances (measured as shortest-path lengths) and pairwise distances in the embedding space. The intuition behind this metric is that, if node features preserve relevant structural information, nodes that are close in the graph should also be close in the embedding space, resulting in a positive correlation. However, we note that this correlation primarily reflects how well embeddings preserve overall topological proximity, rather than directly capturing hierarchical structure, which often involves more nuanced, multi-level dependencies not fully described by shortest-path distances alone. Nevertheless, we expect that *node2vec* and *kamada-kawai* embeddings can still capture certain aspects of hierarchy.

The Spearman correlation was computed individually for each graph in the dataset and then averaged across all graphs. The results are reported in Table 3. As expected, *random* node features yield correlations close to zero, confirming the absence of meaningful structural information. In contrast, *node2vec* embeddings exhibit strong positive correlations, indicating that they effectively preserve local and global graph proximity. Finally, *Kamada-Kawai* features, which are explicitly designed to preserve graph-theoretic distances in Euclidean space, achieve the highest correlations among all feature types.

To further assess the degree to which *node2vec* and *kamada-kawai* features reflect hierarchical structure specifically, we compute the Spearman correlation between each node's depth in the tree (i.e., its level in the hierarchy) and its embedding norm (i.e., distance from the origin in the embedding space). Again, this analysis is performed on a per-graph basis and results are averaged. The results, reported in Table 4, show a consistent positive correlation between node depth and embedding norm, indicating that deeper nodes tend to be embedded farther from the origin. This trend suggests that both *node2vec* and *kamada-kawai* features capture aspects of the hierarchical structure, with a notably stronger alignment observed for *kamada-kawai*.

Task	**Node Feature**	**Euclidean**	**Hyperbolic**
Easy	random	-0.0050 ± 0.0288	-0.0070 ± 0.0407
Easy	node2vec	0.6922 ± 0.0630	0.7244 ± 0.0616
Easy	kamada-kawai	0.9754 ± 0.0128	0.8627 ± 0.0264
Hard	random	-0.0014 ± 0.0352	-0.0012 ± 0.0492
Hard	node2vec	0.7723 ± 0.0251	0.8188 ± 0.0208
Hard	kamada-kawai	0.9656 ± 0.0108	0.8559 ± 0.0212

Table 3: Spearman correlation between graph-theoretic distances and pairwise node embedding distances, computed in Euclidean and hyperbolic space. Results are shown for two tasks (*Easy* and *Hard*) and two node feature types: *random* (Gaussian noise) and *node2vec* embeddings with $(p = 1, q = 4)$ to bias walks toward local neighborhoods.

Task	**Node Feature**	**Euclidean**	**Hyperbolic**
Easy	node2vec	0.5444 ± 0.1558	0.5442 ± 0.1559
Hard	node2vec	0.7150 ± 0.0783	0.7142 ± 0.0780
Easy	kamada-kawai	0.6788 ± 0.2927	0.6788 ± 0.2927
Hard	kamada-kawai	0.8550 ± 0.1574	0.8550 ± 0.1574

Table 4: Spearman correlation between node hierarchy levels (depth from the root) and node distances from the origin in the embedding space, for both Euclidean and hyperbolic geometries, using *node2vec* and *kamada-kawai* features. Results are shown for the *Easy* and *Hard* tasks. Higher values indicate stronger alignment between node depth and embedding position, reflecting better encoding of hierarchical structure.

4 Experiments

We compared the performance of hyperbolic and Euclidean models on our synthetic dataset. As Euclidean baselines, we used two models: the standard Graph Convolutional Network (GCN) [8], which we refer to as Euclidean*, and a variant that incorporates attention weights computed according to Eq.(3), but using Euclidean rather than hyperbolic distances; we refer to this model as Euclidean. The Euclidean node embeddings follow the same processing pipeline shown in part 3 of Fig.1, with the key difference that the Distance Layer is implemented in Euclidean space instead of hyperbolic space. In all models, the resulting graph embeddings are passed through a fully connected layer that applies a linear transformation to produce logits for the two output classes.

We tuned training and architecture hyperparameters separately for the Euclidean and hyperbolic models using a combination of random and grid search strategies. Full details of the hyperparameter settings and search space are provided in Appendix C.

We evaluated the models using embeddings with a comparable number of learnable parameters. Specifically, we used a 2D hyperboloid manifold embedded in $\mathbb{R}^3$ for the hyperbolic model and a 3D Euclidean space for the Euclidean model. This choice ensures both models have similar representational capacity in terms of parameter count: the Euclidean GCN with attention has 1329 parameters, the Euclidean* GCN has 1407, and the fully hyperbolic model (HyboNet) has 1422.

Task	**Geometry**	**Feature**	**Accuracy(%)**	**ROC-AUC(%)**	**F1 Score(%)**	**Precision(%)**	**Recall(%)**
Easy	Hyperbolic	random	96.1 ± 8.0	96.1 ± 8.0	96.0 ± 8.2	96.2 ± 7.5	96.1 ± 8.0
Easy	Euclidean	random	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**
Easy	Euclidean*	random	99.4 ± 0.9	99.4 ± 0.9	99.4 ± 0.9	99.4 ± 0.9	99.4 ± 0.9
Easy	Hyperbolic	node2vec	99.9 ± 0.5	99.9 ± 0.5	99.9 ± 0.5	99.9 ± 0.5	99.9 ± 0.5
Easy	Euclidean	node2vec	**99.9 ± 0.1**	**99.9 ± 0.1**	**99.9 ± 0.1**	**99.9 ± 0.1**	**99.9 ± 0.1**
Easy	Euclidean*	node2vec	98.6 ± 1.4	98.6 ± 1.4	98.6 ± 1.4	98.6 ± 1.4	98.6 ± 1.4
Easy	Hyperbolic	kamada-kawai	89.4 ± 12.2	89.4 ± 12.1	89.4 ± 12.2	89.5 ± 12.0	89.4 ± 12.2
Easy	Euclidean	kamada-kawai	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**	**100.0 ± 0.0**
Easy	Euclidean*	kamada-kawai	91.8 ± 4.5	100.0 ± 0.0	91.8 ± 4.5	91.8 ± 4.5	91.8 ± 4.5
Hard	Hyperbolic	random	76.8 ± 6.2	76.8 ± 5.8	76.6 ± 6.3	77.5 ± 5.8	76.8 ± 6.2
Hard	Euclidean	random	**87.9 ± 12.6**	**87.9 ± 12.6**	**87.8 ± 12.6**	**87.9 ± 12.6**	**87.9 ± 12.6**
Hard	Euclidean*	random	64.4 ± 14.4	64.4 ± 14.4	64.2 ± 14.4	64.5 ± 14.4	64.4 ± 14.4
Hard	Hyperbolic	node2vec	86.9 ± 5.3	86.9 ± 5.3	86.9 ± 5.3	87.0 ± 5.4	86.9 ± 5.3
Hard	Euclidean	node2vec	**99.0 ± 0.9**	**99.0 ± 0.9**	**99.0 ± 0.9**	**99.0 ± 0.9**	**99.0 ± 0.9**
Hard	Euclidean*	node2vec	74.9 ± 1.0	74.9 ± 1.0	74.8 ± 1.0	74.9 ± 0.9	74.9 ± 1.0
Hard	Hyperbolic	kamada-kawai	77.7 ± 19.3	77.7 ± 19.3	77.7 ± 19.3	77.7 ± 19.3	77.7 ± 19.3
Hard	Euclidean	kamada-kawai	**99.9 ± 0.2**	**99.9 ± 0.2**	**99.9 ± 0.2**	**99.9 ± 0.2**	**99.9 ± 0.2**
Hard	Euclidean*	kamada-kawai	60.9 ± 4.3	60.9 ± 4.3	60.1 ± 4.6	61.9 ± 4.8	60.9 ± 4.3

Table 5: Classification performance of Hyperbolic and Euclidean Graph Convolutional Networks on the synthetic dataset for the *Easy* and *Hard* classification tasks. Node features tested include *random* vectors and *node2vec* embeddings ($p = q = 4$). Euclidean models were evaluated both with and without attention mechanisms. The model without attention (standard GCN [8]) is indicated by an asterisk (*) as Euclidean*. Reported metrics are mean ± standard deviation over 11 random seeds and include Accuracy, ROC-AUC, F1 Score, Precision, and Recall. The overall best results are highlighted in bold.

5 Results

Table 5 reports the classification performance across synthetic tasks of two difficulty levels (*Easy* and *Hard*), using three node feature types (*random*, *node2vec*, and *kamada-kawai*). We compare a fully hyperbolic GNN (see Section 2) with two Euclidean baselines described in Section 4: a standard GCN (denoted Euclidean*) and a GCN augmented with an attention mechanism using Euclidean distances (denoted Euclidean). All metrics (accuracy, ROC-AUC, F1 score, precision, and recall) are are reported as mean ± standard deviation over 10 random seeds.

Overall, the Euclidean model consistently outperforms the hyperbolic model across all configurations. In the *Easy* task, both geometries achieve high accuracy, even when node features are uninformative (i.e., *random*). In this case, models rely primarily on structural information from the graph topology, and the Euclidean model with attention achieves the best performance.When using *node2vec* features, both hyperbolic and Euclidean models with attention perform exceptionally well, although the hyperbolic model appears slightly less robust. A surprising result emerges with the *kamada-kawai* features, which are the most hierarchical (as shown in Table 4): these features are leveraged effectively by the Euclidean models, but not by the hyperbolic model which downgrades its performances. In fact, the hyperbolic model performs worse with *kamada-kawai* features than with random features.

In the more challenging *Hard* task, the Euclidean model with attention again outperforms all others. The Euclidean* GCN struggles, confirming the need for attention to capture subtle structural variations. The hyperbolic model remains competitive with *node2vec* features but performs poorly with *kamada-kawai* features.

These results show that a hyperbolic architecture is not necessary for classifying tree-like structures. When using highly hierarchical features, the performance gap between Euclidean and hyperbolic models widens, favoring the former. This counterintuitive outcome likely stems not only from hyperbolic message passing limitations, but also from how features are embedded into hyperbolic space. In particular, hyperbolic models project input features to the tangent space at the origin, followed by an exponential map to embed them into the manifold. As shown by our Spearman correlation analysis in Table 4, this mapping process distorts the hierarchical structure initially encoded in the features. To isolate the effect of this distortion from the influence of the model architecture, we trained a Euclidean model using the hyperbolically mapped *kamada-kawai* features on the *Hard* task. This model achieved an average accuracy of 76.4 ± 2.0 over 10 seeds, lower than when using the original Euclidean features, but still substantially more robust than the corresponding hyperbolic model. Nevertheless, the findings support the conclusion that a hyperbolic architecture is not essential for solving this task, even in the presence of hierarchical graph structures.

6 Limitations

This work can be further strengthened by extending the experimental setup in several directions.

First, our analysis has focused exclusively on 3D node embeddings. Extending the evaluation to a broader range of embedding dimensions would help clarify how the performance gap between hyperbolic and Euclidean models varies with dimensionality, and whether higher-dimensional hyperbolic spaces can better exploit hierarchical signals.

Second, it would be valuable to explore a fully hyperbolic pipeline, in which all operations, including pooling and the final classification layers (see Fig. 1), are performed directly in hyperbolic space, avoiding the need for a transition to Euclidean space via a hyperbolic distance layer.

7 Conclusions

In this work, we conducted a systematic evaluation of a hyperbolic graph neural network in a controlled graph classification setting. We introduced a synthetic benchmark dataset composed of tree-like graphs, where class distinctions arise from variations in branching structure at different hierarchical levels. This setup enabled explicit control over the hierarchical content of the task.

Our results reveal that hyperbolic GNNs do not outperform their Euclidean counterparts, even in an "ideal" setting with purely tree-like structures. Across both *Easy* and *Hard* tasks, Euclidean models—particularly those with attention mechanisms—consistently achieved higher performance, regardless of node feature type.

A particularly surprising finding is that hierarchical node features (e.g., *kamada-kawai*), expected to align well with hyperbolic geometry, were effectively exploited by Euclidean models but degraded performance in hyperbolic ones. We traced this effect to the distortion introduced by the feature mapping pipeline used in HGNNs, which projects features to a tangent space and then applies the exponential map. Our Spearman correlation analysis showed that this process weakens the hierarchical signal encoded in the features.

To isolate the effect of feature distortion from the influence of model architecture, we trained a Euclidean model on the hyperbolically mapped *kamada-kawai* features. While this model performed significantly worse than the original Euclidean baseline in terms of average accuracy, it still outperformed the hyperbolic model in terms of stability, exhibiting much lower variance. These results suggest that, in this case, the observed performance gap stems also from how node features are embedded into hyperbolic space, rather than from fundamental limitations of the hyperbolic architecture itself.

Overall, our findings challenge the assumption that hyperbolic architectures are inherently better suited for graph classification with tree-like (i.e. hyperbolic) structures. We highlight the importance of evaluating not only the latent space geometry, but also how node features interact with that geometry. Our benchmark provides a controlled setting to test such interactions. The results demonstrate that a well-tuned Euclidean GCN can outperform the hyperbolic counterpart, even when the underlying data is highly hierarchical. We leave the advantages of hyperbolic models to unsupervised settings where the goal is to embed graphs with minimal distortion, or to supervised tasks in which labels are explicitly tied to the preservation of distances in the embedding space.

References

1. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. science **286**(5439), 509–512 (1999)
2. Chami, I., Ying, Z., Ré, C., Leskovec, J.: Hyperbolic graph convolutional neural networks. Advances in neural information processing systems **32** (2019)
3. Chen, W., Han, X., Lin, Y., Zhao, H., Liu, Z., Li, P., Sun, M., Zhou, J.: Fully hyperbolic neural networks. arXiv preprint arXiv:2105.14686 (2021)
4. Erdos, P., Rényi, A., et al.: On the evolution of random graphs. Publ. math. inst. hung. acad. sci **5**(1), 17–60 (1960)
5. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)
6. John, M.L.: Introduction to smooth manifolds. Spinger (2012)
7. Katsman, I., Gilbert, A.: Shedding light on problems with hyperbolic graph learning. arXiv preprint arXiv:2411.06688 (2024)
8. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
9. Krioukov, D., Papadopoulos, F., Kitsak, M., Vahdat, A., Boguná, M.: Hyperbolic geometry of complex networks. Physical Review E—Statistical, Nonlinear, and Soft Matter Physics **82**(3), 036106 (2010)
10. Law, M., Liao, R., Snell, J., Zemel, R.: Lorentzian distance learning for hyperbolic representations. In: International Conference on Machine Learning. pp. 3672–3681. PMLR (2019)
11. Lee, J.M.: Riemannian manifolds: an introduction to curvature, vol. 176. Springer Science & Business Media (2006)
12. Lensink, K., Peters, B., Haber, E.: Fully hyperbolic convolutional neural networks. Research in the Mathematical Sciences **9**(4), 60 (2022)
13. Liu, Q., Nickel, M., Kiela, D.: Hyperbolic graph neural networks. Advances in neural information processing systems **32** (2019)
14. Nickel, M., Kiela, D.: Poincaré embeddings for learning hierarchical representations. Advances in neural information processing systems **30** (2017)
15. Nickel, M., Kiela, D.: Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In: International conference on machine learning. pp. 3779–3788. PMLR (2018)
16. Qu, E., Zou, D.: Hyperbolic convolution via kernel point aggregation. arXiv preprint arXiv:2306.08862 (2023)
17. Sansone, P.F.C., Vento, M., Cordella, L., Foggia, P., Sansone, C., Vento, M.: An improved algorithm for matching large graphs. In: Proc. of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations. vol. 57 (2001)
18. Sarkar, R.: Low distortion delaunay embedding of trees in hyperbolic plane. In: International symposium on graph drawing. pp. 355–366. Springer (2011)
19. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world'networks. nature **393**(6684), 440–442 (1998)
20. Zhang, Y., Wang, X., Shi, C., Liu, N., Song, G.: Lorentzian graph convolutional networks. In: Proceedings of the web conference 2021. pp. 1249–1261 (2021)

A Hyperbolic Geometry

In this section, we summarize key concepts from Riemannian geometry, with a focus on definitions and operations in hyperbolic spaces that are relevant to the hyperbolic graph neural network model considered in this work. For a more comprehensive treatment, we refer the reader to [6, 11].

Riemannian Manifold A Riemannian manifold is a pair $(\mathcal{M}, g)$, where $\mathcal{M}$ is a smooth n-dimensional manifold and g is a Riemannian metric on $\mathcal{M}$. That is, around each point $x \in \mathcal{M}$, there exists a neighborhood that is homeomorphic to an open subset of $\mathbb{R}^n$, endowing $\mathcal{M}$ with a topological manifold structure. Moreover, at each point x, the tangent space $T_x\mathcal{M}$ is a real n-dimensional vector space, and hence isomorphic to $\mathbb{R}^n$ as a vector space. The Riemannian metric g is a smooth collection of inner products $g_x \colon T_x\mathcal{M} \times T_x\mathcal{M} \to \mathbb{R}$, one for each $x \in \mathcal{M}$, which varies smoothly with x. This structure allows for the definition of geometric notions such as angles, lengths of curves, geodesic distances, volumes, and curvature.

Geodesics and Induced Distance Function Let $\gamma : [\alpha, \beta] \to \mathcal{M}$ be a smooth curve on a Riemannian manifold $(\mathcal{M}, g)$. The length of γ, defined with respect to the Riemannian metric g, is given by

$$L(\gamma) = \int_\alpha^\beta \|\gamma'(t)\|_g \, dt,$$

where $\|\gamma'(t)\|_g = \sqrt{g_{\gamma(t)}(\gamma'(t), \gamma'(t))}$. The geodesic distance between two points $u, v \in \mathcal{M}$ is defined as

$$d_{\mathcal{M}}(u, v) = \inf_\gamma L(\gamma),$$

where the infimum is taken over all smooth curves γ such that $\gamma(\alpha) = u$ and $\gamma(\beta) = v$.

Exponential and Logarithmic Maps, and Parallel Transport For a point $x \in \mathcal{M}$ and a tangent vector $v \in T_x\mathcal{M}$, there exists a unique geodesic $\gamma : [0, 1] \to \mathcal{M}$ satisfying $\gamma(0) = x$ and $\gamma'(0) = v$. The exponential map at x is defined as

$$\exp_x : T_x\mathcal{M} \to \mathcal{M}, \quad \exp_x(v) = \gamma(1).$$

Its local inverse, where defined, is the *logarithmic map*

$$\log_x : \mathcal{M} \to T_x\mathcal{M}.$$

Given two points $x, y \in \mathcal{M}$, the *parallel transport* operator

$$P_{x \to y} : T_x\mathcal{M} \to T_y\mathcal{M}$$

transports vectors along a geodesic from x to y, preserving the inner product induced by the metric g.

Hyperbolic Space Hyperbolic space is a Riemannian manifold of constant negative curvature. Several equivalent models exist to describe it, including the Poincaré ball, Klein, and Lorentz (hyperboloid) models. In this work, we adopt the Lorentz model due to its closed-form expressions and numerical stability in optimization.

Lorentz Model For a fixed negative curvature $c < 0$, the Lorentz model (also known as the hyperboloid model) is defined as the Riemannian manifold $\left(\mathcal{L}_c^n, g^{\mathcal{L}}\right)$, where

$$\mathcal{L}_c^n = \left\{ x \in \mathbb{R}^{n+1} \,\middle|\, \langle x, x \rangle_{\mathcal{L}} = \frac{1}{c},\ x_0 > 0 \right\},$$

and the Lorentzian inner product $\langle \cdot, \cdot \rangle_{\mathcal{L}}$ is given by

$$\langle x, y \rangle_{\mathcal{L}} = -x_0 y_0 + \sum_{i=1}^{n} x_i y_i, \tag{4}$$

for $x = (x_0, x_1, \dots, x_n),\ y = (y_0, y_1, \dots, y_n) \in \mathbb{R}^{n+1}$. The metric tensor $g^{\mathcal{L}}$ at each point $x \in \mathcal{L}_c^n$ is induced by restricting this Lorentzian inner product to the tangent space $T_x \mathcal{L}_c^n$.

Exponential Map For a point $x \in \mathcal{L}_c^n$ and a tangent vector $v \in T_x \mathcal{L}_c^n$, the exponential map is defined as

$$\exp_x(v) = \cosh\left(\sqrt{-c}\, \|v\|_{\mathcal{L}}\right) x + \sinh\left(\sqrt{-c}\, \|v\|_{\mathcal{L}}\right) \frac{v}{\sqrt{-c}\, \|v\|_{\mathcal{L}}}, \tag{5}$$

where $\|v\|_{\mathcal{L}} = \sqrt{\langle v, v \rangle_{\mathcal{L}}}$ is the Lorentz norm.

Center of Mass in the Lorentz Model Given a set of points $\mathcal{P} = \{x_1, \dots, x_{|\mathcal{P}|}\} \subset \mathcal{L}_c^n$ in the Lorentz model, with associated weights $\{\nu_1, \dots, \nu_{|\mathcal{P}|}\}$, the center of mass is defined as the point $\mu \in \mathcal{L}_c^n$ that minimizes the expected squared Lorentzian distance:

$$\mu = \arg\min_{y \in \mathcal{L}_c^n} \sum_{i=1}^{|\mathcal{P}|} \nu_i \, d_{\mathcal{L}}^2(x_i, y),$$

where the squared Lorentzian distance is defined as

$$d_{\mathcal{L}}^2(a, b) = \frac{2}{c} - 2\langle a, b \rangle_{\mathcal{L}}.$$

This formulation was first introduced by Law et al. [10], who showed that the centroid with respect to the squared Lorentzian distance has a closed-form solution:

$$\mu = \frac{\sum_{i=1}^{|\mathcal{P}|} \nu_i x_i}{\sqrt{-c} \left\| \sum_{i=1}^{|\mathcal{P}|} \nu_i x_i \right\|_{\mathcal{L}}^2}. \tag{6}$$

This notion of centroid is particularly useful in hyperbolic graph learning for tasks such as attention-based message aggregation and class mean computation in hyperbolic space.

B Synthetic Dataset

We propose a dataset generation framework for constructing tree-like structures with high flexibility in task difficulty for graph classification. This flexibility stems from the ability to control several key parameters: the branching factor ranges at each level of the tree, the mutation mechanism applied to branching factors, the mutation rate, and the total number of nodes (which also influences the tree depth). While two task settings — *Easy* and *Hard* — were introduced in Section 3, here we demonstrate that even more challenging configurations can be constructed. Specifically, we propose an *Extra Hard* setting that can serve as a benchmark for future experiments requiring minimal structural bias.

In the *Extra Hard* setting, both classes share the same base branching scheme, defined by the following list of level-wise ranges:

$$[(4,7),\ (4,6),\ (4,5),\ (3,4),\ (3,4),\ (1,2)],$$

with the range $(1,2)$ applied to all subsequent levels until the graph reaches the desired number of nodes. To assign class labels, we inject subtle, localized mutations. For class A, we apply a mutation at level 2, replacing the original branching range with $(2,3)$ for each parent node with probability 0.11. For class B, mutations occur at levels 3 and 4, where the branching factor is altered to $(1,2)$ with probability 0.07. These minimal and targeted perturbations result in nearly indistinguishable structural statistics across classes, pushing the classification task toward a highly non-trivial regime. Fig.(3) shows histograms of the average graph-level metrics computed for each class, illustrating how closely aligned their structural properties have become. A Mann–Whitney U test revealed no statistically significant difference in diameter and average shortest path length, while other metrics (i.e., depth, radius, and degree entropy) exhibited statistically significant but minor differences.

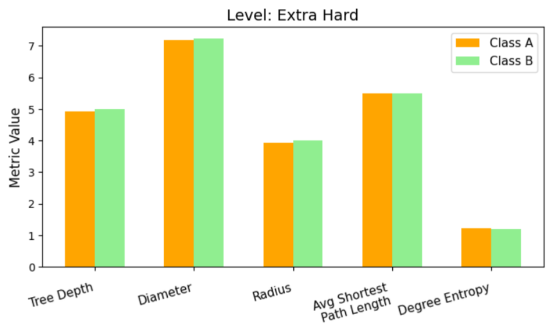

Fig. 3: Histogram of average graph-level metrics for each class in the *Extra Hard* setting.

C Hyperparameter Tuning

This appendix section provides a detailed overview of the hyperparameter tuning process. Hyperparameters were optimized separately for the Euclidean and hyperbolic models, and for each classification task and node feature type individually. To identify the best configurations, we employed a combination of random and grid search strategies over a predefined search space.

The specific hyperparameter ranges considered for training included learning rates, weight decay, and batch sizes. For model-specific parameters, we evaluated various activation functions, weight initialization methods, numbers of learnable centroids, and network depths, as detailed below.

For the training configuration, we searched over learning rates $[5\times10^{-3}, 1\times10^{-3}, 5\times10^{-4}, 1\times10^{-4}, 5\times10^{-5}, 1\times10^{-5}]$, weight decay values $[10^{-2}, 10^{-4}, 10^{-6}]$, and batch sizes $[10, 50, 100]$. For model-specific settings, we explored activation functions {`ReLU`, `LeakyReLU`, `Tanh`, `ELU`}, weight initialization strategies {`Kaiming`, `Uniform`, `xavier`}, number of learnable centroids $[50, 100, 200]$ and number of layers $[1, 2, 3]$.

Table 6: Hyperparameters for Hybonet model with different node features and task difficulties.

	random		node2vec		kamada-kawai	
Task	Easy	Hard	Easy	Hard	Easy	Hard
Learning Rate	5e-4	5e-4	5e-4	5e-4	5e-4	5e-4
Weight Decay	1e-6	1e-6	1e-6	1e-6	1e-6	1e-6
Batch Size	100	100	100	100	100	100
Activation	LeakyReLU	LeakyReLU	LeakyReLU	LeakyReLU	LeakyReLU	LeakyReLU
Weight Initialization	Uniform	Uniform	Uniform	Uniform	Uniform	Uniform

Table 7: Hyperparameters for GCN model w/ att with different node features and task difficulties.

	random		node2vec		kamada-kawai	
Task	Easy	Hard	Easy	Hard	Easy	Hard
Learning Rate	5e-3	5e-3	5e-3	5e-3	5e-3	5e-3
Weight Decay	1e-4	1e-2	1e-4	1e-6	1e-4	1e-6
Batch Size	50	100	100	50	50	50
Activation	LeakyReLU	Tanh	LeakyReLU	LeakyReLU	LeakyReLU	Tanh
Weight Initialization	Uniform	Uniform	Uniform	Uniform	Uniform	Uniform

Table 8: Hyperparameters for GCN model with different node features and task difficulties.

	random		node2vec		kamada-kawai	
Task	Easy	Hard	Easy	Hard	Easy	Hard
Learning Rate	5e-3	5e-3	5e-3	5e-3	5e-3	5e-3
Weight Decay	1e-6	1e-6	1e-6	1e-6	1e-6	1e-6
Batch Size	10	50	10	50	10	10
Activation	ReLU	LeakyReLU	Tanh	ReLU	ReLU	ReLU
Weight Initialization	Xavier	Uniform	Kaiming	Uniform	Xavier	Uniform