Geometry-Aware Edge Pooling for Graph Neural Networks

Katharina Limbeck[*,1,2], Lydia Mezrag[*,3,4], Guy Wolf[†,3,4], and Bastian Rieck[†,1,2,5]

[1] Helmholtz Munich, Ingolstädter Landstraße 1, 85764 Neuherberg, Germany
[2] Technical University of Munich, Arcisstraße 21, 80333 Munich, Germany
[3] Université de Montréal, 2900 Édouard-Montpetit, Montréal QC H3T 1J4, Canada
[4] Mila - Quebec AI Institute, 6666 Rue Saint-Urbain, Montréal QC H2S 3H1, Canada
[5] Université de Fribourg, Av. de l'Europe 20, 1700 Fribourg, Switzerland
[*] These authors contributed equally. [†] These authors jointly supervised this work.

Abstract. Graph Neural Networks (GNNs) have shown significant success for graph-based tasks. Pooling layers are crucial components of GNNs that enable faster training and potentially better generalisation by reducing the size of input graphs. However, existing pooling operations often optimise for the learning task at the expense of discarding fundamental graph structures, thus reducing interpretability. This leads to unreliable performance across varying dataset types, downstream tasks and pooling ratios. Addressing these concerns, we propose novel graph pooling layers for structure-aware pooling via edge collapses. Our methods leverage diffusion geometry and iteratively reduce a graph's size while preserving both its metric structure and structural diversity. We guide pooling using *magnitude*, an isometry-invariant diversity measure, which permits us to control the fidelity of the pooling process. Further, we use the *spread* of a metric space as a faster and more stable alternative ensuring computational efficiency. Empirical results demonstrate that our methods (i) achieve superior performance compared to alternative pooling layers across a range of diverse graph classification tasks, (ii) preserve key spectral properties of the input graphs, and (iii) retain high accuracy across varying pooling ratios.

Keywords: Graph Pooling · Geometric Deep Learning · GNNs

1 Introduction

Graph pooling layers are important components of successful GNN architectures. They are implemented alongside convolutional layers to reduce the size of graph representations enabling GNNs to scale to large and complex real-world graphs. However, the choice of pooling method strongly influences downstream-applications and task-performance. In fact, the question of which graph properties to preserve during pooling [33, 36, 45], remains an ongoing debate. It is thus crucial to design expressive, efficient, and interpretable pooling layers that are capable of reliably encoding task-relevant information. Most graph pooling literature takes a node-centric view [33]. However, this focus on *node-centric* rather than *edge-centric* pooling often leads to the loss of important structural information. Common pooling methods either drop nodes or optimise for a node clustering while treating graph connectivities as a secondary objective. As visualised in 1, this frequently leads to counter-intuitive pooling decisions that fail to retain key geometric structures in the graph. Addressing these concerns, topological and geometric descriptors of graphs are uniquely poised to interoperate structural information into graph pooling. Throughout this work, we treat graphs as metric spaces and assess their geometry via diffusion distances, which naturally work alongside message passing to effectively encode key graph structures. Motivated by ongoing research on novel geometric invariants, we find that generalised measures of size and diversity are especially promising candidates for guiding graph pooling. In particular, we use the *magnitude* of a metric space, which measures a graph's structural diversity, to control the loss of structural information during edge pooling [31]. Our work is motivated by successful applications of magnitude across a range of machine learning tasks, such as the evaluation of diversity for latent spaces [32], boundary detection for images [1], and the study of generalisation for neural networks [2, 3]. Building up on this research, we are the first to propose the use of magnitude in the context of graph learning. We further advance on existing applications by investigating an alternative and closely-related measure, known as the spread of a metric space, to greatly improve the computational efficiency of our methods. Our main **contributions** are as follows:

– We propose MagEdgePool and SpreadEdgePool, novel edge-contraction based pooling layers, which preserve graphs' structural diversity.
– We investigate the spread of a metric space as a faster and more stable alternative to magnitude.
– We show that our methods preserve key structural properties during pooling.
– We find that our algorithms perform well at graph classification, surpassing alternative pooling layers across varying datasets and pooling ratios.

2 Background

2.1 Graph Pooling for Graph Neural Networks

As an ongoing field of interest for graph learning, a wide range of pooling methods has been proposed. The most successful of these methods are hierarchical and sequentially coarsen the graph representation while reducing its size [33]. Research has traditionally focused on *node-based* pooling via node clustering or node dropping. Amongst node drop pooling methods, Node Decimal Pooling (NDP) [7] is non-trainable, while TopK [9, 23] and SAGPool [28], are trainable approaches. However, all these methods inevitably lose information because they remove entire sections of the graph during pooling [33]. Node clustering approaches, are either non-trainable, such as Graclus [14] and Non-Negative Matrix Factorization (NMF) [5], or trainable, such as MinCUT [6] and DiffPool [46], which output dense graph representations. In comparison to node drop pooling, clustering-based methods usually require high memory costs [33]. Trainable methods have the

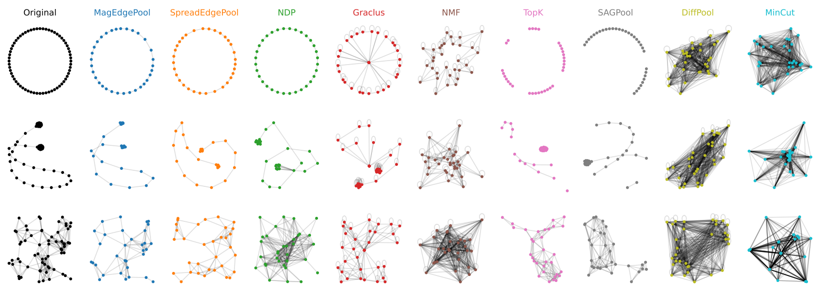


Fig. 1: Pooled graphs across different pooling layers. Our proposed methods, MagEdgePool and SpreadEdgePool, respect the original graphs' geometry during pooling. Alternative approaches create counter-intuitive edges (Graclus, NMF), disconnect entire portions of the graphs (TopK, SAGPool), or return dense representations that do not preserve any geometric structure (DiffPool, MinCut).

potential to better optimise for a specific objective, but might overfit to the task. Non-trainable methods can act as a stronger inductive bias on the underlying graph representation, and do not introduce additional trainable parameters or optimisation objectives [26]. *Edge-based* methods have been studied less extensively than node-centric pooling [15], [27] despite their potential for encoding connectivities in a more faithful manner. EdgePool [15], the most successful edge-based method, uses iterative edge-contraction based on edge scores, which are learned from features of adjacent nodes. However, EdgePool does not explicitly consider a graph's topology and selecting edges based on node features can lead to counter-intuitive decisions, for example by retaining local structures in strongly-connected communities [15]. Interpretability remains a leading concern [33], and non-trainable approaches are capable of being well-performing and traceable pooling methods [26]. We therefore find that there is strong potential for designing geometry-aware edge pooling operations. Figure 1 further illustrates how many of the aforementioned pooling methods inherently fail to preserve key graph structures even for simple toy examples. Trainable methods in this overview were optimised for a spectral loss [26], but this does *not* ensure interpretable preservation of graphs' underlying geometry. Addressing these shortcomings, it is of interest to leverage alternative tools from computational geometry to quantify the qualitative difference between graphs. While pooling based on spectral properties has been investigated extensively [7, 26], alternative geometric invariants like curvature or persistent homology have only been explored more recently [20, 45], and have not yet been applied to edge pooling specifically.

2.2 The Magnitude and Spread of Metric Spaces

Magnitude is an invariant of (finite) metric spaces that measures the 'effective size' of a space. It is a measure of entropy and diversity [31] that has first been proposed in theoretical ecology [40]. Since its mathematical formalisation [29], magnitude has been connected to numerous key geometric invariants, such as entropy, curvature, density, volume, and intrinsic dimensionality [31]. Because of its intriguing theoretical properties magnitude has received increasing interest for machine learning tasks [32]. However, the magnitude of graphs [30], despite being a strong graph invariant, has not yet found its way into applications.

We consider an undirected finite graph $G = (X, E)$ with n nodes as a *finite metric space* consisting of the node set X equipped with a metric $d\colon X \times X \to \mathbb{R}_{\geq 0}$. Its similarity matrix $\zeta_X \subseteq \mathbb{R}^{n\times n}$ is defined by $\zeta_X(x, y) = e^{-d(x,y)}$ for $x, y \in X$. This allows us to introduce similarity-dependent notions of the diversity of metric spaces. To this end, we define a *weighting* on the metric space (X, d), which is a vector $w \in \mathbb{R}^n$ such that $\zeta_X w = \mathbb{1}$, where $\mathbb{1}$ is the column vector of ones. Whenever such a weighting exists, the *magnitude* of the metric space (X, d) is *uniquely* defined by $\mathrm{Mag}(X) = \sum_{i=1}^{n} w(i)$. This is guaranteed if ζ_X is positive definite, which essentially means that ζ_X is invertible. A finite metric space with positive definite similarity matrix is called *positive definite* [35]. Metric spaces of negative type are positive definite [31]; this includes $\mathbb{R}^n$ equipped with Euclidean distances [29], effective resistance distances [13], and diffusion distances [10].

Throughout this paper, we will refer to the magnitude of a graph G as the magnitude of its associated metric space (X, d). That is, we define the *magnitude of a graph* as

$$\mathrm{Mag(G)} = \sum_{x,y\in X} \zeta_X^{-1}(x, y). \tag{1}$$

Closely related to magnitude, the *spread* of a metric space is another measure of 'size' introduced by [42]. Given a metric graph $G = (X, E)$ with the graph metric d, its *spread* is defined by

$$\mathrm{Sp}(G) := \sum_{x\in X} \frac{1}{\sum_{y\in X} e^{-d(x,y)}}. \tag{2}$$

As diversity measures on graphs, both *magnitude* and *spread* summarise the number of distinct sub-communities in a network based on the distance metric and degree of similarity between nodes. This view on structural diversity naturally aligns with our goal of contracting redundant graph structures during pooling. Throughout this work, we investigate to what extent spread is a valid alternative to magnitude in practice. Magnitude is well-studied and has received increasing interest in both theory [31] and application [32]. However, computing it either requires inverting a matrix, solving a system of linear equations [32], or approximating [4], which can be notoriously expensive or unstable to compute. Metric-space *spread* in comparison can be computed given *any* distance (obviating the requirement of metric spaces of negative type), making it much more versatile [42]. Moreover, as the sum of reciprocal mean similarities, spread can be calculated or approximated [18] much more efficiently than magnitude and does not require inverting a matrix. Although spread has been studied less extensively [18, 42], there are strong reasons to assume that it shares the same advantages as magnitude. In fact, for a positive definite metric space X, we have $\mathrm{Sp}(X) \leq \mathrm{Mag}(X)$ [42, Theorem 2]. Moreover, magnitude and spread coincide for finite homogeneous metric spaces [42, Theorem 3], such as the ring graph in Figure 1. In practice, as we further explore in Appendix C.1, the magnitude and spread of graphs from real-world datasets, such as NCI1, exhibit nigh-perfect correlation when computed based on diffusion distances, underlining the strong connection between the two quantities.

2.3 Diffusion Geometry on Graphs

In this work, we use diffusion distances to compute magnitude and spread on graphs. This is motivated by their desirable theoretical properties and the capability of diffusion to aid and act along message passing in GNNs [24].

We now briefly detail the type of diffusion distance used throughout this paper [10]. Consider a graph $G = (X, E)$ and its adjacency matrix A. The latter could be an affinity matrix derived from a kernel. Let D be the diagonal degree matrix whose diagonal entries $D_{ii} = \sum_{j=0}^{n} A_{ij}$ equal the degree of each vertex. The symmetrically *normalised adjacency matrix* $\hat{A} := D^{-\frac{1}{2}} A D^{\frac{1}{2}}$ is a simple example of a *Markov transition matrix* and represents the probability of moving from one vertex to another. The *normalised graph Laplacian* is defined as $\hat{L} = I - \hat{A} = D^{-\frac{1}{2}}(D - A)D^{-\frac{1}{2}}$. Since $\hat{L}$ is symmetric and positive definite it has positive eigenvalues $2 \geq \lambda_0 > \lambda_1 > \lambda_2 > \cdots > \lambda_{N-1} \geq 0$ with eigenvectors $\{\psi_l\}_l$. This provides a natural embedding of the graph G in Euclidean space given by:

$$\Phi(x) = (\lambda_1 \psi_1(x), \cdots, \lambda_{N-1} \psi_{N-1}(x)) \text{ for } x \in X. \tag{3}$$

The *diffusion distance* is then defined by:

$$d(x, y) = |\Phi(x) - \Phi(y)|_2 \text{ for } x, y \in X. \tag{4}$$

Theorem 1. *Any finite metric space (X, d) endowed with the diffusion distance is positive definite.*

Proof. See Appendix A.1.

As a consequence of Theorem 1, the magnitude of any metric graph equipped with this diffusion distance is well defined. Diffusion distances are at comparable scales across graphs because the normalised graph Laplacian, which works with the relative conductivities, is robust to varying node degrees and graph sizes. This enables us to compute and compare magnitude and spread directly.

3 Methods

3.1 Magnitude-Guided Graph Pooling

At the heart of our approach, we use magnitude or spread to monitor and control structural changes in the graph during edge contraction pooling. Edge contraction is chosen as a pooling operation because it respects graph connectivity, while

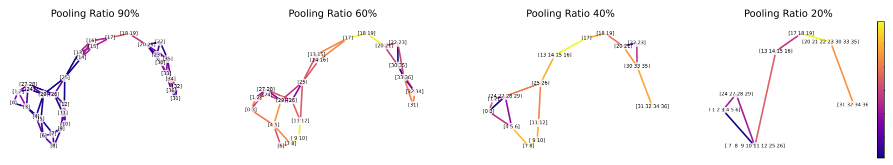


Fig. 2: Illustrating our proposed pooling method, MagEdgePool, on a graph from the ENZYME dataset across varying pooling ratios. Each edge is coloured by its magnitude difference, which measures the impact its contraction would have on the graph's structural diversity. Edges with low magnitude differences are most redundant for the graph's geometry and are collapsed first.

outputting a sparsely-connected graph representation. Conceptually, edge pooling thus aligns well with the goal of making minimal changes to the graph and keeping its diversity and geometry as unchanged as possible. Intuitively, pooled graphs with comparable magnitude will be similar in terms of effective size. That is, the effective number of distinct communities in the graphs are deemed similar based on their diffusion distance, i.e. information-flow between vertices. More formally, let $G = (X, E)$ be a graph and denote by G/e the graph resulting from the contraction of the edge $e \in E$ in G. We choose a pooling ratio $r \in (0, 1]$ and aim to reduce the graph to the corresponding number of nodes i.e. $k = \lfloor r \cdot |X| \rceil$. Initially, we set the pooled graph $G' = (X', E') := G$. To assess which edges to contract first, we determine their importance for the graph's global geometry by computing a selection score for each edge defined as

$$s(e) = |\mathrm{Mag}(G) - \mathrm{Mag}(G/e)|. \tag{5}$$

That is, we calculate the difference in magnitude between the original graph and the graph for which the edge has been collapsed. In this manner, we score an edge's relevance for the graph structure by the impact its collapse would have on the graph's magnitude. At each iteration i, we then select the edge with the lowest magnitude difference

$$e_i \in \arg\min_{e \in E' \setminus E_c} s(e) \tag{6}$$

and assign $G' = G'/e_i$ where $E_c \subseteq E'$ is the set of all edges that are not adjacent to an already contracted edge. The edge to contract e_i is chosen randomly amongst the possible choices whenever there is more than one valid option. If all edges that meet this requirement have been collapsed, but the pooling ratio is not reached, we re-compute the edge scores on the new graph and repeat the same procedure. We stop if the pooling ratio is reached, i.e. $|X'| = \lfloor r \cdot |X| \rceil$, or if there are no edges left in the reduced graph, i.e. $E' = \emptyset$. This approach allows us to flexibly reduce graphs to any desired size.

Edge pooling then gives a hard assignment of nodes, where each vertex is assigned to a single super-node in the output graph based on the neighbours it has been merged with. To compress the node features, we average the features of any node that contributed to a pooled super-node. This ensures that information on all nodes' features is preserved during pooling. Restricting the number of times a vertex can be merged is enforced to prevent our methods from collapsing entire portions of a graph early in the pooling process. This enables a more uniform pooling across the graph, which aids feature preservation and expressivity.

Our methods assume that the most redundant edges will be those whose removal would change the diversity of the graph the least. Magnitude informs us about the global importance of an edge. Intuitively, that means we want to start with merging edges from well connected communities whenever their contraction does not notably change the graph's effective size. As illustrated in Figure 2, this ensures that globally influential edges will be collapsed late in the pooling progress. For example, the edge which bridges the two parts of this enzyme graph, is scored more highly and thus merged later than well-connected cliques. Redundant local structures are collapsed first, and the overall diffusion geometry is respected. For the graph in Figure 2 this ensures that key topological characteristics, such as the cycle, are retained across the pooling process. Our algorithm is explained in more detail in Appendix B.4. Whenever we use magnitude to compute the edge scores, we denote our algorithm by MagEdgePool. Similarly, the use of spread is denoted by SpreadEdgePool.

3.2 Theoretical Analysis

We now present theoretical properties of our pooling methods. First, we highlight fundamental invariances and properties of magnitude and spread used to design our algorithms. Further, we provide a bound on the difference in magnitude during pooling by the difference in spread, demonstrating the close relationship between the two. For a complete list or theorems and proofs see Appendix A.3.

Additivity for disjoint graphs. An important property of magnitude is that it behaves like the cardinality of sets. This behaviour is useful when computing the magnitude of a disconnected graph by splitting the problem into calculating the magnitude of the disconnected components.

Theorem 2. *Consider a positive definite metric graph $G = G_1 \sqcup G_2$ consisting of the disjoint union of two graphs G_1 and G_2. Then $\mathit{Mag}(G) = \mathit{Mag}(G_1) + \mathit{Mag}(G_2)$.*

Isomorphism invariance. Our pooling layers are invariant under isometries of the input graph provided that the edge choice at each iteration is deterministic whenever the edge scores coincide. This is a consequence of the following result.

Theorem 3. *If the metric graphs G_1 and G_2 are isometric then $\mathit{Mag}(G_1) = \mathit{Mag}(G_2)$ and $\mathit{Sp}(G_1) = \mathit{Sp}(G_2)$.*

Edge contraction on graphs. Edge contraction is the main operation of our pooling methods and can be considered as a map f between graphs. The following result provides a sufficient condition to ensure that the morphism f is compatible with the metric structure.

Theorem 4. *Consider an edge-contraction map $f : (G_1, d_1) \to (G_2, d_2)$ between positive definite metric graphs. If the map is Lipschitz (i.e. $d_2(f(v_1), f(v_2)) \leq d_1(v_1, v_2)$ $\forall v_1, v_2 \in X_1$) then $Mag(G_2) \leq Mag(G_1)$.*

Starting from an initial metric graph $G = (X, E)$, a Lipschitz edge-contraction map f yields a sequence of graphs $\{G_i\}_{i=1}^{k}$ where k is the number of edges that have been contracted. This sequence can be constructed as described by our algorithms in Section 3.1. Let $\Delta^{(k)}\mathrm{Mag}(G) = |\mathrm{Mag}(G^{(k-1)}) - \mathrm{Mag}(G^{(k)})|$ and let $\Delta^{(k)}\mathrm{Sp}(G) = |\mathrm{Sp}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)})|$.

Bounding magnitude by spread. We track the difference in magnitude and spread throughout the edge contraction process detailed above. This allows us to propose an inequality that describes the relation between the difference of magnitude and spread during pooling. The bound then demonstrate the close conceptual relationship between SpreadEdgePool and MagEdgePool.

Theorem 5. *Consider a positive definite metric graph G with positive weights. Assume that the edge-contraction maps describing MagEdgePool and SpreadEdgePool induce distance decreasing surjections on the vertex sets. If $|Mag(G^{(k-1)}) - Sp(G^{(k)})| \leq C\Delta^{(k)}Sp(G)$, then*

$$\Delta^{(k)}Mag(G) \leq 3C\Delta^{(k)}Sp(G).$$

Computational Complexity. The time complexity of our pooling methods is independent of the GNN and is dominated by the cost of computing the edge scores in Equation (5). Given a graph $G = (X, E)$, magnitude has time complexity $O(|X|^3)$. In comparison, spread has time complexity $O(|X|^2)$ and can be more efficiently approximated [18]. Spread thus offers a considerably faster alternative. Now, let $O(C_d)$ be the time complexity of computing the metric on G. The time complexity of SpreadEdgePool is dominated by $O(|E|(C_d + |X|^2 + \log|E|))$ and MagEdgePool by $O(|E|(C_d + |X|^3 + \log|E|))$. We note that on large graphs, it is possible to speed up the distance computations further to ensure scalability. See Appendix A.2 for a full description of computational costs and Appendix C.2 for an empirical evaluation, which shows that our algorithm performs on-par with existing pooling methods across the datasets evaluated throughout this work.

4 Experimental Results

Across our experiments, we address the following three key tasks: (i) graph classification performance, (ii) graph structure preservation during pooling, (iii) and performance across varying pooling ratios.

4.1 Graph Classification

We first investigate how well our proposed edge pooling methods perform at graph classification tasks in comparison to alternative pooling layers.

Table 1: Mean and standard deviation of the graph classification accuracy. The best performing model is marked in bold. All models that did not perform significantly different from the best model are coloured green. The rightmost column shows the mean rank of each pooling method across datasets.

Method	ENZYMES	PROTEINS	Mutagenicity	DHFR	IMDB-B	IMDB-M	NCI1	NCI109	Mean Rank
No Pooling	87.3 ± 2.5	73.8 ± 0.8	80.1 ± 1.3	71.4 ± 1.9	69.7 ± 0.7	46.0 ± 0.7	76.5 ± 1.8	74.3 ± 2.0	-
MagEdge	91.5 ± 3.2	76.4 ± 3.9	**77.5 ± 2.7**	88.0 ± 3.8	72.4 ± 1.7	47.4 ± 1.7	72.7 ± 2.4	73.0 ± 3.3	**2.4**
SpreadEdge	**92.8 ± 1.6**	75.1 ± 3.1	76.0 ± 4.0	**90.7 ± 3.8**	71.8 ± 1.5	47.3 ± 1.7	73.4 ± 2.5	71.8 ± 1.8	**3.0**
NDP	92.2 ± 1.6	73.7 ± 3.9	73.4 ± 3.1	79.6 ± 4.4	**73.3 ± 2.0**	47.3 ± 2.5	70.6 ± 2.2	70.0 ± 2.2	3.6
Graclus	91.3 ± 3.7	**76.6 ± 3.7**	72.5 ± 2.0	64.4 ± 5.8	71.9 ± 1.5	**49.3 ± 2.4**	68.8 ± 1.4	69.5 ± 2.1	5.6
NMF	78.6 ± 8.0	73.0 ± 8.1	71.0 ± 4.7	66.5 ± 7.7	69.4 ± 2.5	43.3 ± 1.7	71.0 ± 3.7	72.0 ± 5.3	6.0
TopK	82.2 ± 7.5	73.2 ± 1.4	75.8 ± 4.7	68.9 ± 3.0	68.9 ± 1.5	45.6 ± 1.0	**75.3 ± 2.4**	73.9 ± 3.3	4.9
SAGPool	82.4 ± 4.5	73.8 ± 1.3	76.0 ± 2.6	69.9 ± 3.0	69.1 ± 0.6	45.7 ± 0.5	74.3 ± 2.8	**74.0 ± 2.3**	4.7
DiffPool	74.0 ± 5.7	68.9 ± 2.0	68.4 ± 1.9	79.8 ± 3.2	68.3 ± 0.8	44.4 ± 0.8	68.9 ± 1.0	68.3 ± 1.9	7.6
MinCut	80.2 ± 6.6	75.6 ± 1.3	70.9 ± 1.5	63.8 ± 3.7	69.3 ± 0.7	46.1 ± 0.8	66.7 ± 1.4	66.9 ± 2.0	7.4

Experimental Setup. We evaluate 8 different graph datasets [37], as detailed in Appendix B.2. If node features are unavailable, we use node degree as an input feature. We follow an established experimental benchmark [26] and guidance for fair model comparison [19]. Specifically, we plug in each pooling layer into the model architecture of the following form [26]:

$$\mathrm{MLP}(\mathbf{X}) \rightarrow \mathrm{GNN}(\mathbf{X}, \mathbf{A}) \rightarrow \mathrm{POOL}(\mathbf{X}, \mathbf{A}) \rightarrow \mathrm{GNN}(\mathbf{X}, \mathbf{A}) \rightarrow \mathrm{SUM}(\mathbf{X}) \rightarrow \mathrm{MLP}(\mathbf{X})$$

The model includes pre-processing and post-processing MLPs with 2 layers, 256 hidden units, ReLU activation, and batch normalization. $\mathrm{GNN}(\mathbf{X}, \mathbf{A})$ refers to a graph neural network layer, more specifically a general convolutional layer [47] with parameters chosen according to the best results achieved in a GNN benchmark [47]. As an intermediate layer, $\mathrm{POOL}(\mathbf{X}, \mathbf{A})$ corresponds to a specific pooling layer. All pooling layers are configured to pool each graph to around 50% of nodes. We also compare with 'No Pooling,' the same model architecture without any pooling layers. We use 10-fold stratified cross-validation and further partition the training data into 90% training and 10% validation data while keeping the labels balanced between splits. Finally, we report the best test accuracy of each model trained using Adam with a cross-entropy loss (batch size 32, learning rate 0.0005, and early stopping based on the validation loss with a patience of 50 epochs). Further details are described in Appendix B.5.

Classification Results. Table 1 reports the mean and standard deviation of the test accuracy achieved by different pooling methods. We furthermore highlight which methods do *not* perform statistically significantly different from the best model (using pairwise Wilcoxon signed-rank tests applied to the accuracy scores and employing Holm–Bonferroni correction at a significance threshold of $p = 0.05$) to identify pooling methods that achieve top performance.

Notably, both MagEdgePool and SpreadEdgePool achieve the *best mean ranks* across datasets in terms of their accuracy. Further, they are always among the top-performing methods across all evaluated datasets. Altogether, both their ranking and their individual accuracy scores thus demonstrate superior and consistently high performance across graph classification tasks.

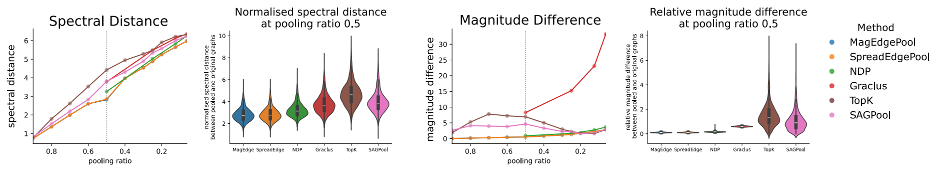


Fig. 3: Structure preservation for all graphs in the NCI1 dataset across varying pooling ratios. Left: The spectral distance between the normalised Laplacians of the the original and the pooled graphs. Right: The relative difference in magnitude, which summarises the proportional difference in structural diversity after pooling. Violinplots show the variability across graphs at pooling ratio 0.5.

The performance benefits of our pooling methods are most pronounced on DHFR [41], where they surpass even the GNN without pooling layer by around 17 percentage points. This provides evidence that the regularising effects of our pooling approach can help reduce overfitting, especially for small datasets and geometrically-rich graphs. For other biological datasets (Mutagenicity, NCI1 and NC109), our methods show competitive performance with trainable layers, indicating that the introduction of additional trainable components into the pooling layer is *not* necessary to guarantee high task performance. On ENZYMES, DHFR, IMDB-BINARY, and IMDB-MULTI, non-trainable methods generally outperform trainable pooling layers, with MagEdgePool and SpreadEdgePool consistently reaching high accuracy.

Comparing pooling methods to using no pooling, we observe that MagEdgePool and SpreadEdgePool reach similar or even higher performance across datasets. For six datasets, our diversity-guided pooling methods improve mean accuracy, indicating that pooling retains task-relevant information while aiding the generalisation capabilities of the GNN. MagEdgePool and SpreadEdgePool act as interpretable and expressive pooling transformations, which reduce the computational costs making GNNs learn from graphs' coarsened geometry.

As reported in Table 1, MagEdgePool and SpreadEdgePool achieve very similar accuracies across datasets superior to alternative pooling layers. In practice, especially for large graphs, we recommend using SpreadEdgePool due to its high predictive performance and superior computational efficiency.

4.2 Magnitude and Graph Structure Preservation

Motivated by the visual comparison of graphs pooled using different pooling layers from Figure 1, we next set out to investigate the link between structure preservation and task performance. Specifically, we choose NCI1, a dataset of 4,110 graphs corresponding to chemical compounds, because it has been shown to possess both informative features and task-relevant graph structures [11]. We follow the same classification procedure as before and extract the pooled graph representations after training the GNNs described in Section 4.1. Three pooling layers, MinCut, DiffPool and NMF, were removed from further comparison because they showed notably worse qualitative results for the motivating examples in Figure 1 and classification performance in Table 1. NDP and Graclus are evaluated across fewer pooling ratios than more adaptive methods, because they pool graphs to around half their size at every step. To assess graph structure preservation we use the spectral distance defined as $\sqrt{\sum_{k=1}^{K}(\lambda_k - \lambda'_k)^2}$, the l_2-norm between the eigenspectra of the normalised Laplacians of the original and the pooled graphs [43]. We also report the magnitude difference between graphs to evaluate the preservation of structural diversity.

MagEdgePool and SpreadEdgePool show low spectral distances across pooling ratios as visualised in Figure 3. This indicates that contracting edges guided by structural diversity preserves key spectral properties. Our methods also demonstrate low magnitude differences confirming that they perform as intended. In fact, the structure preservation scores for MagEdgePool and SpreadEdgePool coincide almost perfectly, giving empirical evidence that spread offers an alterative to magnitude. Figure 3 indicates that preserving magnitude corresponds to lower spectral distances and better retention of spectral properties. This link supports our motivation of guiding pooling by magnitude.

Alternative pooling layers fail to effectively preserve graphs' structural properties during both qualitative and quantitative comparisons to varying extents. Node decimal pooling (NDP) [7] was specifically designed to preserve spectral properties during pooling. However, it still reaches both higher spectral distances and higher magnitude differences than MagEdgePool on average across pooling ratios as visualised in Figure 3. Finally, the sparse pooling layers Graclus, TopKPool, and SAGPool, all show higher spectral distances than our approach. This difference is even more pronounced in terms of magnitude differences, where all these three methods demonstrate high distortion of the underlying metric space diversity. These findings agree with the qualitative comparisons between graphs pooled using different pooling layers as visualised in Figure 1. We thus conclude that MagEdgePool and SpreadEdgePool successfully encode graphs' coarsened geometry during pooling, surpassing alternative pooling methods.

4.3 Pooling Ratio and Task Performance

From Table 1 we observe that it is possible to reach very high performance on benchmark datasets even while pooling each graph to half its size. Based on this, we further investigate how the pooling ratio influences pooling layers and their classification performance. We consider two datasets, NCI1 and ENZYMES, which contains 600 graphs representing protein tertiary structures from 6 classes of enzymes and is selected as an example of a multi-class prediction task. We keep the same experimental setup described in Section 4.1, but use GIN layers instead of general convolutional layers to further assess whether the trends in performance differ across model architectures. Figure 4 then reports the classification accuracy achieved by each pooling layer for varying pooling ratios.

Notably, we observe that MagEdgePool and SpreadEdgePool consistently reach very high test accuracies even at low pooling rations. Meanwhile, the

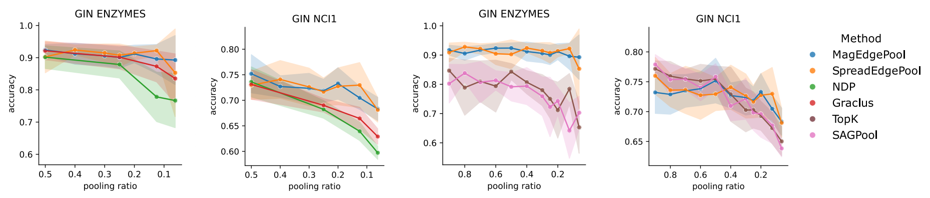


Fig. 4: Classification performance across varying the pooling ratio for different pooling layers. Pooling is applied as part of a GIN architecture. Results are shown for the ENZYME and NCI1 datasets. Lines show the mean and shaded areas the standard deviation of the test accuracy.

performance of other non-trainable methods drops notably more when graphs are pooled to up to 6.25% of their original size showing that they fail to preserve task-relevant information for both ENZYMES and NCI1. We note that the trainable pooling layers, TopK and SAGPool, reach higher or comparable performance on NCI1 for pooling ratios above 50%, but decrease in accuracy for more extreme pooling ratios. They consistently perform worse on ENZYMES, indicating that they distort important graph features or key graph structures during pooling. MagEdgePool and SpreadEdgePool in comparison reach superior performance and lower decreases in accuracy across varying pooling ratios demonstrating their potential to offer reliable, interpretable and stable pooling operations. Overall, the reported accuracies in Figure 4 agree with results in Table 1 indicating that our observations hold for varying choices of GNN layers. We thus find across experiments that MagEdgePool and SpreadEdgePool constitute useful general-purpose pooling approaches, demonstrating their capability to faithfully encoding graphs' geometry, which ensures stable performance across pooling ratios, datasets and GNN architectures.

5 Discussion

Our pooling methods implicitly require redundancy and homophily in the graph representation and assume that preserving graph structure is *beneficial* to the learning task. Moreover, our algorithm relies on efficient distance computations and we only explore one case of diffusion distances, which could be generalised further in future work. Although our methods are non-trainable, our experiments show that trainable pooling layers do not guarantee higher performance.

Across experiments we find that MagEdgePool and SpreadEdgePool constitute useful general-purpose pooling approaches that perform well for a wide range of classification tasks and pooling ratios. Guiding edge pooling to preserve graphs' structural diversity successfully encodes key graph properties and ensures stable and interpretable performance. Further, we overcome one major limitation of computing magnitude on large graphs by proposing the spread of a metric space as a faster and closely-related alternative, which has the potential to aid research in geometric deep learning beyond the scope of this paper.

Bibliography

[1] Adamer, M.F., De Brouwer, E., O'Bray, L., Rieck, B.: The magnitude vector of images. Journal of Applied and Computational Topology **8**(3), 447–473 (2024)

[2] Andreeva, R., Dupuis, B., Sarkar, R., Birdal, T., Simsekli, U.: Topological generalization bounds for discrete-time stochastic optimization algorithms. Advances in Neural Information Processing Systems **37**, 4765–4818 (2024)

[3] Andreeva, R., Limbeck, K., Rieck, B., Sarkar, R.: Metric space magnitude and generalisation in neural networks. In: Topological, Algebraic and Geometric Learning Workshops 2023. pp. 242–253. PMLR (2023)

[4] Andreeva, R., Ward, J., Skraba, P., Gao, J., Sarkar, R.: Approximating metric magnitude of point sets. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 39, pp. 15374–15381 (2025)

[5] Bacciu, D., Di Sotto, L.: A non-negative factorization approach to node pooling in graph convolutional neural networks. In: AI* IA 2019–Advances in Artificial Intelligence: XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19–22, 2019, Proceedings 18. pp. 294–306. Springer (2019)

[6] Bianchi, F.M., Grattarola, D., Alippi, C.: Spectral clustering with graph neural networks for graph pooling. In: International conference on machine learning. pp. 874–883. PMLR (2020)

[7] Bianchi, F.M., Grattarola, D., Livi, L., Alippi, C.: Hierarchical representation learning in graph neural networks with node decimation pooling. IEEE Transactions on Neural Networks and Learning Systems **33**(5), 2195–2207 (2020)

[8] Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**, i47–i56 (2005)

[9] Cangea, C., Veličković, P., Jovanović, N., Kipf, T., Liò, P.: Towards sparse hierarchical graph classifiers. arXiv preprint arXiv:1811.01287 (2018)

[10] Coifman, R.R., Lafon, S.: Diffusion maps. Applied and computational harmonic analysis **21**(1), 5–30 (2006)

[11] Coupette, C., Wayland, J., Simons, E., Rieck, B.: No metric to rule them all: Toward principled evaluations of graph-learning datasets. arXiv preprint arXiv:2502.02379 (2025)

[12] David, G., Averbuch, A.: Hierarchical data organization, clustering and denoising via localized diffusion folders. Applied and Computational Harmonic Analysis **33**(1), 1–23 (2012)

[13] Devriendt, K., Lambiotte, R.: Discrete curvature on graphs from the effective resistance. Journal of Physics: Complexity **3**(2), 025008 (2022)

[14] Dhillon, I.S., Guan, Y., Kulis, B.: Weighted graph cuts without eigenvectors a multilevel approach. IEEE transactions on pattern analysis and machine intelligence **29**(11), 1944–1957 (2007)

[15] Diehl, F.: Edge contraction pooling for graph neural networks. arXiv preprint arXiv:1905.10990 (2019)
[16] Diehl, F., Brunner, T., Le, M.T., Knoll, A.: Towards graph pooling by edge contraction. In: ICML 2019 workshop on learning and reasoning with graph-structured data (2019)
[17] Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. Journal of molecular biology **330**(4), 771–783 (2003)
[18] Dunne, K.: Efficiently approximating spread dimension with high confidence. arXiv preprint arXiv:2408.14590 (2024)
[19] Errica, F., Podda, M., Bacciu, D., Micheli, A., et al.: A fair comparison of graph neural networks for graph classification. In: Proceedings of the Eighth International Conference on Learning Representations (ICLR 2020) (2020)
[20] Feng, A., Weber, M.: Graph pooling via ricci flow. arXiv preprint arXiv:2407.04236 (2024)
[21] Fey, M., Lenssen, J.E.: Fast graph representation learning with pytorch geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
[22] Fey, M., Sunil, J., Nitta, A., Puri, R., Shah, M., Stojanovic, B., Bendias, R., Alexandria, B., Kocijan, V., Zhang, Z., He, X., Lenssen, J.E., Leskovec, J.: PyG 2.0: Scalable learning on real world graphs. In: Temporal Graph Learning Workshop, KDD (2025)
[23] Gao, H., Ji, S.: Graph u-nets. In: Proceedings of the 36th International Conference on Machine Learning (2019)
[24] Gasteiger, J., Weißenberger, S., Günnemann, S.: Diffusion improves graph learning. Advances in neural information processing systems **32** (2019)
[25] Grattarola, D., Alippi, C.: Graph neural networks in tensorflow and keras with spektral. IEEE Computational Intelligence Magazine **16**(1), 99–106 (2021)
[26] Grattarola, D., Zambon, D., Bianchi, F.M., Alippi, C.: Understanding pooling in graph neural networks. IEEE transactions on neural networks and learning systems **35**(2), 2708–2718 (2022)
[27] Landolfi, F.: Revisiting edge pooling in graph neural networks. In: ESANN (2022)
[28] Lee, J., Lee, I., Kang, J.: Self-attention graph pooling. In: International conference on machine learning. pp. 3734–3743. pmlr (2019)
[29] Leinster, T.: The magnitude of metric spaces. Documenta Mathematica **18**, 857–905 (2013)
[30] Leinster, T.: The magnitude of a graph. In: Mathematical Proceedings of the Cambridge Philosophical Society. vol. 166, pp. 247–264. Cambridge University Press (2019)
[31] Leinster, T.: Entropy and Diversity: The Axiomatic Approach. Cambridge University Press (2021)
[32] Limbeck, K., Andreeva, R., Sarkar, R., Rieck, B.: Metric space magnitude for evaluating the diversity of latent representations. In: Advances in Neural Information Processing Systems. vol. 37, pp. 123911–123953 (2024)

[33] Liu, C., Zhan, Y., Wu, J., Li, C., Du, B., Hu, W., Liu, T., Tao, D.: Graph pooling for graph neural networks: progress, challenges, and opportunities. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence. pp. 6712–6722 (2023)

[34] Long, A.W., Ferguson, A.L.: Landmark diffusion maps (l-dmaps): Accelerated manifold learning out-of-sample extension. Applied and Computational Harmonic Analysis **47**(1), 190–211 (2019)

[35] Meckes, M.W.: Positive definite metric spaces. Positivity **17**(3), 733–757 (2013)

[36] Mesquita, D., Souza, A., Kaski, S.: Rethinking pooling in graph neural networks. Advances in Neural Information Processing Systems **33**, 2220–2231 (2020)

[37] Morris, C., Kriege, N.M., Bause, F., Kersting, K., Mutzel, P., Neumann, M.: Tudataset: A collection of benchmark datasets for learning with graphs. arXiv preprint arXiv:2007.08663 (2020)

[38] Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., Schomburg, D.: Brenda, the enzyme database: updates and major new developments. Nucleic acids research **32**, D431–D433 (2004)

[39] Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(9) (2011)

[40] Solow, A.R., Polasky, S.: Measuring biological diversity. Environmental and Ecological Statistics **1**, 95–103 (1994)

[41] Sutherland, J.J., O'brien, L.A., Weaver, D.F.: Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. Journal of chemical information and computer sciences **43**(6), 1906–1915 (2003)

[42] Willerton, S.: Spread: a measure of the size of metric spaces. International Journal of Computational Geometry & Applications **25**(03), 207–225 (2015)

[43] Wills, P., Meyer, F.G.: Metrics for graph comparison: a practitioner's guide. Plos one **15**(2), e0228728 (2020)

[44] Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1365–1374 (2015)

[45] Ying, C., Zhao, X., Yu, T.: Boosting graph pooling with persistent homology. Advances in Neural Processing Systems (2024)

[46] Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. Advances in neural information processing systems **31** (2018)

[47] You, J., Ying, Z., Leskovec, J.: Design space for graph neural networks. Advances in Neural Information Processing Systems **33**, 17009–17021 (2020)

Acknowledgments. This work was partially funded by the Helmholtz International Lab Causal Cell Dynamics InterLabs-0029 grant by the Initiative and Networking Fund of the Hermann von Helmholtz-Association Deutscher Forschungszentren e.V. [K.L.], Mila EDI scholarships [L.M.], Humboldt Research Fellowship, CIFAR AI Chair, NSERC Discovery grant 03267, FRQNT grant 343567, and NSF grant DMS-2327211 [G.W.]. This work has received funding from the Swiss State Secretariat for Education, Research, and Innovation (SERI). K.L. is supported by the Helmholtz Association under the joint research school 'Munich School for Data Science (MUDS).' The content provided here is solely the responsibility of the authors and does not necessarily represent the views of the funding agencies.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Appendices and Supplementary Material

To elaborate on the results reported in our main paper, we first detail extended theoretical results and proofs for our theoretical contributions. Next, we detail our experimental evaluation, the assets used for our experiments, and the algorithm describing our pooling methods. Finally, we report extended results on the experiments included in our main paper.

A Theoretical Analysis

This section details full proofs and extended explanations for the mathematical theory introduced in Section 2 and the theoretical analysis of our pooling methods described in Section 3.2.

A.1 Diffusion Distances

As detailed in Section 2.3, the *diffusion distance* is defined by:

$$d(x,y) = |\Phi(x) - \Phi(y)|_2 \text{ for } x, y \in X. \tag{7}$$

Theorem 1 *Any finite metric space (X, d) endowed with the diffusion distance is positive definite.*

Proof. By definition of the diffusion distance, the map $\Phi : X \to \mathbb{R}^{N-1}$ in Equation (3) defines an isometry $(X, d) \hookrightarrow l_2^{N-1} := (\mathbb{R}^{N-1}, d_2)$, where d_2 is the metric induced by the l_2-norm. Finally, by Theorem 2.5.3 in [29], subsets of Euclidean space l_2^{N-1} are positive definite.

A.2 Computational Complexity

We next analyse the computational complexity of our pooling methods, which are described in Appendix B.4 and Section 3.1. Specifically, we expand on the statements in Section 3.2 by detailing the computational complexity of the pooling process. Given a graph $G = (X, E)$, let $k = \lfloor (1-r)|X| \rfloor$ be the number of nodes that should be contracted as determined by the pooling ratio r. The time complexity of our pooling approach can be split up into the following steps:

Computing magnitude or spread. Magnitude has time complexity $O(|X|^3)$ and can further be approximated via iterative normalisation in $O(i \times |S_i| \times |X|^2)$ time assuming G has a positive weighting where i is the number of iterations and $S_i \subset X$ [4]. Spread computations have time complexity $O(|X|^2)$, which is a notable improvement to magnitude. It is possible to approximate spread computations via subsets [18] or iterative optimisation using mini-batching [4]. For i iterations on subsets $S_i \subset X$, the time complexity of approximating spread reduces to $O(i \times |S_i| \times |X|)$ [4, 18]. Spread thus offers a much faster alternative to magnitude and can scale to large graphs considerably more efficiently.

Computing distances and similarities. For large datasets, it is key to speed up the distance calculations. Diffusion distances have time complexity $O(|X|^3)$, but can be reduced to $O(k|X|^2)$ when restricting the computations to the top k eigenvectors. Diffusion maps can further be approximated via low-rank approximations. To reduce the cost of repeated distance computations, it is possible to approximate the metric on the reduced graph G/e by directly updating the distances for G. Given a distance matrix, computing the similarity matrix then has linear time complexity in the number of entries. Denote the time complexity of computing the distances and similarities by $O(C_d)$.

Edge contraction. To get $G' = G/e$, contracting an edge $e \in E$ takes $O(|X|)$ time.

Edge score computations. For each edge, its edge score is computed by applying the edge contraction and computing the magnitude or spread of the reduced graph, which takes $O(|X| + C_d + C_S)$ time, where C_S refers to the cost of computing either magnitude or spread as detailed above. Note that the computation of these edge scores is independent across edges and can be parallelised.

Edge score sorting. The edge scores can be sorted from lowest to highest in $O(|E| \log |E|)$ time.

Feature aggregation. The node features, $\mathbf{F} \subseteq \mathbb{R}^{|X|,F}$, can be aggregated in $O(|X| \times F)$ time.

Putting this all together, we get that the cost of our pooling algorithms can be described by a worst-case time complexity of

$$O(|E|(|X| + C_d + C_S + log|E|) + |X|(F + k))$$

if $k \leq 0.5|X|$ and the graph is not pooled to less than half its size. Otherwise, if $k > 0.5|X|$, we re-compute the edge scores whenever no valid edges are left as described in Section 3.1. In this scenario, the first term of the complexity expression is repeated, corresponding to re-computations on successively smaller graphs. In summary, the overall time complexity of our pooling method is dominated by the cost of calculating and sorting the edge scores. This cost is independent of the choice of GNN architecture, ensuring that the training costs remain stable and do not escalate with model complexity. In practice, as further explored in Appendix C.2, we thus find that our pooling algorithms perform on par with alternative pooling layers in terms of computational efficiency.

A.3 Magnitude and Spread

Additivity for disjoint graphs As a measure of the effective size, one appealing property of magnitude is that it behaves akin to cardinality. In fact, magnitude is additive when taking the disjoint union of multiple metric spaces.

Theorem 2 *Consider a positive definite metric graph $G = G_1 \sqcup G_2$ consisting of the disjoint union of two graphs G_1 and G_2. Then $Mag(G) = Mag(G_1) + Mag(G_2)$.*

Proof. Let $G = (X, E)$ together with the metric d be a positive definite metric graph. Assume G is a disjoint union of two metric graphs (G_1, d_1) and (G_2, d_2) over the vertex sets X_1 and X_2, respectively, such that $d_{|X_1} = d_1$ and $d_{|X_2} = d_2$ and $d(x_1, x_2) = \infty$ for all $x_1 \in X_1$, $x_2 \in X_2$. Then, $\zeta_G = \zeta_{G_1} \oplus \zeta_{G_2}$ and $\mathrm{Mag}(G) = \mathrm{Mag}(G_1) + \mathrm{Mag}(G_2)$.

This result applies to graphs equipped with the shortest-path distance or the diffusion distance considered in this paper, because the distance between two nodes depends only on the connected component they belong to and is infinite if there is no path between them. Therefore, we can naturally see the similarity matrix ζ_G as block-diagonal and compute the magnitude of G by summing up the magnitude of its disconnected subgraphs.

Isomorphism invariance A key property of magnitude and spread is that they are isometry invariants of metric spaces. Note that by *graph isometry* we mean an isometry on the underlying vertex set equipped with a metric.

Theorem 3 *If the metric graphs G_1 and G_2 are isometric then $Mag(G_1) = Mag(G_2)$ and $Sp(G_1) = Sp(G_2)$.*

Proof. Let $f : (X_1, d_1) \to (X_2, d_2)$ be a bijective isometry between the metric graphs G_1 and G_2, respectively. Then, for all $x, y \in X_1$ we have that $d_2(f(x), f(y)) = d_1(x, y)$. A consequence of the bijectivity of f is that the distance matrices coincide (up to permutations) and that $\zeta_1 = \zeta_2$. This implies in turn that $\mathrm{Mag}(G_1) = \mathrm{Mag}(G_2)$ and that $\mathrm{Sp}(G_1) = \mathrm{Sp}(G_2)$.

Based on this property for magnitude and spread, we can show that isometry invariance also holds for our proposed pooling algorithm further detailed in Appendix B.4.

Corollary 1 *MagEdgePool and SpreadEdgePool are isometry-invariant if applied to isomorphic graphs provided the choice of edges to contract at each iteration is deterministic whenever the edge scores coincide.*

Proof. Let $f : (G_1, d_1) \to (G_2, d_2)$ be a graph isomorphism and an isometry. From Theorem 3 we have that the edge scores as defined in Equation (5) for $e_1 \in E_1$ and $f(e_1) \in E_2$ will coincide, i.e. $s(e_1) = s(f(e_1))$. Because the choice of edges to contract at each iteration is further assumed to be deterministic if edge scores coincide, it follows that every edge to contract in G_2 corresponds to the image $f(e)$ of an edge e to contract in G_1 and vice versa. Hence, the pooled graphs output by our algorithm are isomorphic.

Edge contraction on graphs We will first recall important results about magnitude in the context of (strictly) positive definite finite metric spaces and refer the interested reader to work by Leinster [29] for further details.

Proposition 1 (Proposition 2.4.3, [29]) *Let (X, d) be a positive definite metric space with finite cardinality $|X| = n$. Then*

$$\mathit{Mag}(X) = \sup_{v \in \mathbb{R}^n \setminus \{0\}} \frac{(\sum_{i=0}^{n} v_i)^2}{v^t \zeta_X v}. \tag{8}$$

Proposition 1 implies that the magnitude of positive definite metric spaces is always positive. Another consequence of this result is a *monotonicity* property on subsets of these metric spaces.

Corollary 2 (Corollary 2.4.4, [29]) *Let (X, d) be a positive definite finite metric space and consider a subset $Y \subset X$ (endowed with the induced metric). Then*

$$\mathit{Mag}(Y) \leq \mathit{Mag}(X). \tag{9}$$

We will now show an analogous result for graphs constructed via edge contraction.

Theorem 4 *Consider an edge-contraction map $f : (G_1, d_1) \to (G_2, d_2)$ between positive definite metric graphs. If the map is Lipschitz (i.e. $d_2(f(v_1), f(v_2)) \leq d_1(v_1, v_2)\ \forall v_1, v_2 \in X_1$) then $\mathit{Mag}(G_2) \leq \mathit{Mag}(G_1)$.*

Proof. Let $f : (G_1, d_1) \to (G_2, d_2)$ be an edge-contraction map and let $n := |X_1|$ and $m := |X_2|$. We will identify $\mathbb{R}^m$ with a subset of $\mathbb{R}^n = \mathbb{R}^m \oplus \mathbb{R}^{n-m}$ using the map $(v_1, \cdots, v_m) \in \mathbb{R}^m \hookrightarrow (v_1, \cdots, v_m, 0, \cdots, 0) \in \mathbb{R}^n$. Then,

$$\begin{aligned} v^t \zeta_{X_1} v &= \sum_{i,j} v_i \zeta_{X_1}[i,j] v_j \\ &= \sum_{i,j} v_i \big(e^{-d_1(x_i, x_j)}\big) v_j \\ &\leq \sum_{i,j} v_i \zeta_{X_2}[i,j] v_j, \end{aligned}$$

and

$$\frac{1}{v^t \zeta_{X_2} v} \leq \frac{1}{v^t \zeta_{X_1} v} \ \forall \ v \neq 0. \tag{10}$$

Finally, by Proposition 1 and Inequality 10, we get that

$$\begin{aligned}\mathrm{Mag}(G_1) = \sup_{v \in \mathbb{R}^n \setminus \{0\}} \frac{(\sum_{i=0}^n v_i)^2}{v^t \zeta_{X_1} v} &\geq \sup_{v \in \mathbb{R}^m \setminus \{0\}} \frac{(\sum_{i=0}^n v_i)^2}{v^t \zeta_{X_1} v} \\ &\geq \sup_{v \in \mathbb{R}^m \setminus \{0\}} \frac{(\sum_{i=0}^m v_i)^2}{v^t \zeta_{X_2} v} = \mathrm{Mag}(G_2)\end{aligned}$$

Bounding magnitude by spread Recall that for positive definite metric spaces, magnitude is known to be an upper bound for spread.

Theorem 6 (Theorem 2.2. [42]). *Suppose that X is a finite metric space. If X is positive definite then*

$$Sp(X) \leq Mag(X).$$

We will now use this bound as well as the results in Appendix A.3 to investigate the relationship between MagEdgePool and SpreadEdgePool. Through a process of iterated edge contraction, our pooling algorithm produces a sequence of hierarchically pooled graphs (as described in Section 3.1 and Appendix B.4). Note that it is not guaranteed that MagEdgePool and SpreadEdgePool yield the same sequence. For this reason, we will refer to the graphs resulting from the k^{th} edge-contraction with magnitude and spread by $G^{(k)}$ and $\widetilde{G}^{(k)}$ respectively.

For each k the edge contraction map $G^{(k)} \to G^{(k+1)}$ is a surjection on the underlying vertex sets $X^{(k)}$ and $X^{(k+1)}$ respectively, i.e $X^{(k+1)} \subset X^{(k)}$. Moreover, we will assume that for any k this map is distance-decreasing. That is, $d^{(k+1)}(f(x_i), f(x_j)) \leq d^{(k)}(x_i, x_j)$ for all $x_i, x_j \in X^{(k+1)}$.

Recall that for any k, $\mathrm{Mag}(G^{(k)})$ is the magnitude of a finite positive definite metric space $(X^{(k)}, d^{(k)})$. Then, by Theorem 4, we deduce that for any k,

$$\mathrm{Mag}(G^{(k+1)}) \leq \mathrm{Mag}(G^{(k)}) \tag{11}$$

Note that by construction, scoring the edges in Algorithm B.4 translates into the following:

$$X^{(k)} = \mathrm{argmin}_{Y \subset X^{(k-1)}, |Y|+1=|X^{(k-1)}|} |\mathrm{Mag}(X^{(k-1)}) - \mathrm{Mag}(Y)| \tag{12}$$

and,

$$\widetilde{X}^{(k)} = \mathrm{argmin}_{Y \subset \widetilde{X}^{(k-1)}, |Y|+1=|\widetilde{X}^{(k-1)}|} |\mathrm{Sp}(\widetilde{X}^{(k-1)}) - \mathrm{Sp}(Y)|. \tag{13}$$

Then, by monotonicity of magnitude (Inequality 11),

$$X^{(k)} = \mathrm{argmax}_{Y \subset X^{(k-1)}, |Y|+1=|X^{(k-1)}|} \mathrm{Mag}(Y) = \mathrm{argmax}_{Y \subset X^{(k-1)}} \mathrm{Mag}(Y).$$

Let $\varDelta^{(k)}\mathrm{Mag}(G) = |\mathrm{Mag}(G^{(k-1)}) - \mathrm{Mag}(G^{(k)})|$ and let $\varDelta^{(k)}\mathrm{Sp}(G) = |\mathrm{Sp}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)})|$. For the following result, we assume that scores are only computed once and that they are the only criterion for edge contraction. Furthermore, we will assume that spread is monotonically decreasing.

Theorem 5 *Consider a positive definite metric graph G with positive weights. Assume that the edge-contraction maps describing MagEdgePool and SpreadEdgePool induce distance decreasing surjections on the vertex sets. If $|Mag(G^{(k-1)}) - Sp(G^{(k)})| \leq C\Delta^{(k)} Sp(G)$, then*

$$\Delta^{(k)} Mag(G) \leq 3C\Delta^{(k)} Sp(G).$$

Proof. For any k we have the following inequality:

$$|\mathrm{Mag}(G^{(k-1)}) - \mathrm{Mag}(G^{(k)})| \leq |\mathrm{Sp}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)})| + |\mathrm{Mag}(G^{(k)}) - \mathrm{Sp}(G^{(k)})| + |\mathrm{Mag}(G^{(k-1)}) - \mathrm{Sp}(G^{(k-1)})|.$$

Assume that $|\mathrm{Mag}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)})| \leq C\Delta^{(k)}\mathrm{Sp}(G)$ for some constant $C > 0$. By the monotonicity of magnitude (Theorem 4), we get that $\mathrm{Mag}(G^{(k)}) \leq \mathrm{Mag}(G^{(k-1)})$ and,

$$\mathrm{Mag}(G^{(k)}) - \mathrm{Sp}(G^{(k)}) \leq \mathrm{Mag}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)}) \leq C\Delta^{(k)}\mathrm{Sp}(G).$$

Similarly, assuming monotonicity of spread yields $\mathrm{Sp}(G^{(k)}) \leq \mathrm{Sp}(G^{(k-1)})$ and,

$$\mathrm{Mag}(G^{(k-1)}) - \mathrm{Sp}(G^{(k-1)}) \leq \mathrm{Mag}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)}) \leq C\Delta^{(k)}\mathrm{Sp}(G).$$

Since $0 \leq \mathrm{Sp}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)}) \leq \mathrm{Mag}(G^{(k-1)}) - \mathrm{Sp}(G^{(k)})$, the constant C must be greater than or equal to 1. We conclude that

$$\Delta^{(k)}\mathrm{Mag}(G) \leq 3C\Delta^{(k)}\mathrm{Sp}(G).$$

B Extended Methods

B.1 Hardware and Software

The experiments reported in our study were implemented using `spektral 1.3.1` [25][1], and `tensorflow 2.16.2`[2]. As further detailed in Appendix B.5 and Section 4.1, we base our graph classification experiments on the benchmark setup and code by [26][3], which also include implementations for the pooling layers compared across our study. By relying on this existing framework, we aim to ensure the reproducibility of our results.

Further, to calculate magnitude and spread we rely on `magnipy`, a Python package [32][4] for magnitude and diversity computations. As a novel contribution of this paper, we extend the computation of magnitude to graph data and novel

[1]https://graphneural.network/ available under an MIT license.

[2]https://pypi.org/project/tensorflow/2.16.2/ available under the Apache Software License (Apache 2.0).

[3]https://github.com/danielegrattarola/SRC available to the research community [26].

[4]https://github.com/aidos-lab/magnipy available under a BSD 3-Clause License.

Table 2: Summary of the graph datasets considered for our experiments.

dataset	library	# classes	# node features	# graphs	avg # nodes	avg # edges
MUTAG	TUDataset	2	no	187	18	40
Enzymes	TUDataset	6	18	600	33	62
COX2	TUDataset	2	3	467	41	44
DHFR	TUDataset	2	3	756	42	45
IMDB-B	TUDataset	2	no	1000	20	97
IMDB-M	TUDataset	3	no	1500	13	65
AIDS	TUDataset	2	4	2000	15	16
Proteins	TUDataset	2	29	1113	39	72
Mutagenicity	TUDataset	2	no	4337	30	31
NCI1	TUDataset	2	no	4110	30	32
NCI109	TUDataset	2	no	4127	30	32
OGBG-MOLHIV	OGB	2	9	41127	25	27
BZR	TUDataset	2	3	405	36	38
BZR_MD	TUDataset	2	no	306	21	225
COX2_MD	TUDataset	2	no	303	26	335
DHFR_MD	TUDataset	2	no	393	24	283
ER_MD	TUDataset	2	no	446	21	235

graph metrics. Specifically, we modify the computations, so that the magnitude of disconnected subgraphs is computed separately (based on Theorem 2) using the `NetworkX`[5] package. We also implement graph distances that have not previously been used to compute magnitude, such as the diffusion distances detailed in Section 2.3. All experiments were run requesting one GPU with 32GB video memory or less.

B.2 Datasets

We briefly describe the graph datasets analysed throughout our work. Simulated graphs, as used for Figure 1, are created using either `PyGSP` [6] or `NetworkX`[7] and all example graphs were created to consist of 64 nodes. For our main graph classification experiments, we analyse six graph datasets taken from biological or chemical applications [8, 17, 38, 39, 41], and two datasets which represent social networks [44]. All datasets are taken either from the `TUDataset`[8] benchmark [37] or the Open Graph Benchmark[9]. More specifically, the main results in Table 1 analyse the following graph classification datasets described in Table 2. Note that we only consider node and not edge features for our experiments.

B.3 Magnitude and Spread Computations

Across our experiments, we compute magnitude and spread as outlined in the main text, and further described in Appendix B.1. Elaborating on these descriptions, we now aim to give an extended explanation of practical and theoretical considerations for computing the magnitude of graphs in practice.

[5]https://github.com/networkx/networkx available under a BSD 3-Clause License.
[6]https://pygsp.readthedocs.io/en/stable/ available under a BSD-3-Clause license.
[7]https://github.com/networkx/networkx available under a BSD 3-Clause License.
[8]https://chrsmrrs.github.io/datasets/ available under a CC BY 4.0 license.
[9]https://ogb.stanford.edu/ available under an MIT licence.

Defining the magnitude of a graph. In mathematical literature, the magnitude of graphs is often studied with the shortest path metric [30]. However, shortest path distances are not guaranteed to be of negative type, thus leading to scenarios and well-known examples for which the similarity matrix is not invertible and magnitude based on this metric cannot be computed [29]. In comparison, resistance distances, diffusion distances, or Euclidean distance always permit the computation of magnitude. Because of this difference in the choice of distance metric, we note that our definition of the magnitude of a graph in Section 2.2 differs from the definition used by Leinster [30]. While we choose to investigate diffusion distances, we note that the distance metric can easily be replaced if needed to explore alternative geometries.

Diffusion geometry. For further research, we believe that the usage of diffusion distances offers the chance to leverage a rich theory on approximation methods via landmarks [34], or localised diffusion computations [12], which can lead to further computational improvements and extension of our methods.

Magnitude and spread as multi-scale functions. Note that magnitude and spread can also be defined as multi-scale functions i.e. $t \mapsto \mathrm{Mag}((X, t \cdot d))$ for a metric space (X, d) and a scale parameter $t \in \mathbb{R}^+$. This parameter t can be likened to choosing a kernel bandwidth or the scale of distances or similarity determining when observations are considered to be distinct. In practical applications, it is advisable to carefully consider the choice of scaling factor t or the type of normalisation used to compare distances [32]. [32] propose a heuristic that uses root-finding to find a suitably large t. However, this requires repeated computations of magnitude and increases the computational costs. A faster and more desirable default choice of t would be based solely on the distance metric. For diffusion distances, we find that setting $t = 1$ is sufficient for our goals. This is because, as discussed in Section 2.3, diffusion distances are computed from the normalised graph Laplacians and are inherently comparable across graphs. Nevertheless, investigating magnitude and spread as multi-scale functions on graphs remains an interesting extension for further work.

Magnitude and spread as diversity measures. We extensively discuss the relationship between magnitude and spread throughout our work. However, our main paper does not have the space to fully explain the theoretical motivations behind the formulations of magnitude, spread, and other generalised measures of diversity. For a more complete discussion we refer the interested reader to [31], an extensive reference work on the mathematical motivation behind entropy and diversity. Furthermore, [42] specifically discusses the spread of a metric space, and [32] describe the practical usage of magnitude as a diversity measure in ML. These works also give descriptions on how and why the magnitude or spread of a metric space can be interpreted as an effective size i.e. as the effective number of distinct points in a metric space or the number of dissimilar nodes in a graph.

B.4 Pooling Algorithm

We now detail our pooling algorithm introduced in Section 3.1 by describing a pseudocode implementation. Note that to describe the algorithm we assume we have pre-selected a distance metric for computing either magnitude or spread.

Algorithm 1 Graph Pooling Methods: SpreadEdgePool and MagEdgePool

```
Require: input graph $G = (X, E)$, node features $\mathbf{F} \in \mathbb{R}^{n \times d}$, pooling ratio $r \in (0, 1]$,
    diversity measure $\mathrm{Mag}(G)$ or $\mathrm{Sp}(G)$
Ensure: Pooled graph $G' = (X', E')$, pooled features $\mathbf{F}'$
 1: Initialise the super-node set $\mathcal{S}(x) \leftarrow x$ for all $x \in X$
 2: Initialise the set of contracted edges $E_c \leftarrow \emptyset$
 3: Initialise the pooled graph $G' \leftarrow G$
 4: Compute initial edge scores:
        $$s(e) = |\mathrm{Mag}(G) - \mathrm{Mag}(G/e)| \quad \forall e \in E$$
 5: while $|X'| \neq \lfloor r|X| \rceil$ AND $|E'| \neq \emptyset$ do
 6:     Select edge $e = (x, y) = \arg\min_{e \in E \setminus E_c} s(e)$
 7:     if $e$ is not adjacent to any previously contracted edge in $E_c$ then
 8:         Contract edge $e$, update $G' \leftarrow G'/e$
 9:         Add $e$ to $E_c$
10:         Update the node selection: merge $\mathcal{S}(x)$ and $\mathcal{S}(y)$
11:     end if
12:     if no more valid edges AND pooling ratio not reached then
13:         Recompute the edge scores $s(e)$ on the updated graph $G'$
14:         Reset $E_c \leftarrow \emptyset$
15:     end if
16: end while
17: Initialize $\mathbf{F}' \leftarrow \emptyset$
18: for each supernode representative $w \in \mathcal{S}$ do
19:     Let $S_w = \{x \in X \mid \mathcal{S}(x) = w\}$
20:     Compute aggregated feature:
            $$\mathbf{F}'_{w,:} = \frac{1}{|S_w|} \sum_{x \in S_w} \mathbf{F}_{x,:}$$
21:     Append $\mathbf{F}'_w$ to $\mathbf{F}'$
22: end for
23: return Pooled graph $G'$, pooled features $\mathbf{F}'$
```

B.5 Extended Experimental Details

We briefly describe extended details for our main experiments.

Overview Experiment To create the illustration in Figure 1 and visually compare the outputs of different pooling methods, we follow the experimental setup by [26] on understanding structure preservation in graph pooling layers. Specifically, we simulate a ring graph with 64 nodes, a barbell graph with 20 nodes on each side connected by 24 nodes in the middle, and a sensor graph with 64 nodes. Graphs are then pooled to a pooling ratio of approximately 50%. All trainable pooling layers were then trained in a self-supervised manner to optimise the following spectral loss between the original graph G and the pooled graph G':

$$\mathcal{L}(G, G') = \sum_{i=0}^{d} F_{:,i}^{\top} L F_{:,i} - F_{:,i}^{\prime\top} L' F'_{:,i} \tag{14}$$

where L, L' are the corresponding Laplacian matrices. The features F are taken to be the top 10 eigenvectors of L concatenated with the coordinates of the nodes in G. F' is the reduced version of F after pooling. Note that this type of spectral loss is one particular proposal on structure preservation and alternative objectives could be investigated.

Structure Preservation Experiment and Pooling Ratios For reporting the structure preservation results described in Section 4.2, we considered multiple different proposals for what it means to preserve graph structure during pooling. In the end, we settled to compare the spectral distance between the symmetrically normalised graph Laplacians as an established measure of spectral property preservation, and investigated the relative difference in magnitude between the original graph G and the pooled graph G' computed from diffusion distances. Specifically, the reported relative magnitude difference is calculated as

$$\mathrm{MagDiff}(G, G') = \frac{|\mathrm{Mag}(G) - \mathrm{Mag}(G')|}{\mathrm{Mag(G)}}. \tag{15}$$

For this experiment we further varied the pooling ratios across different pooling methods. However, some of the pooling layers considered in our study (NDP, Graclus and NMF) were configured to always pool graphs to around half their size. To allow us to compare these methods across increasing pooling ratios, we choose to reapply these pooling operations repeatedly, which is why these three pooling methods are evaluated at pooling ratios that are powers of 0.5.

Graph Classification Experiment Our graph classification architecture follows the experimental setup described in Section 4.1 and is based the benchmark [26]. Across our main classification experiment detailed in Section 4.1, different pooling layers are configured to reduce each input graph to around 50% of the number of nodes in the original graphs. Depending on the pooling method, this is chosen so each graph is reduced to 50% of its original size $k = \lfloor 0.5 * N \rfloor$, (for NDP, Graclus, MagEdgePool, SpreadEdgePool, TopKPool, and SAGPool), or to 50% the average size of all graphs in the training dataset $k = \lfloor 0.5 * \bar{N} \rfloor$ (for

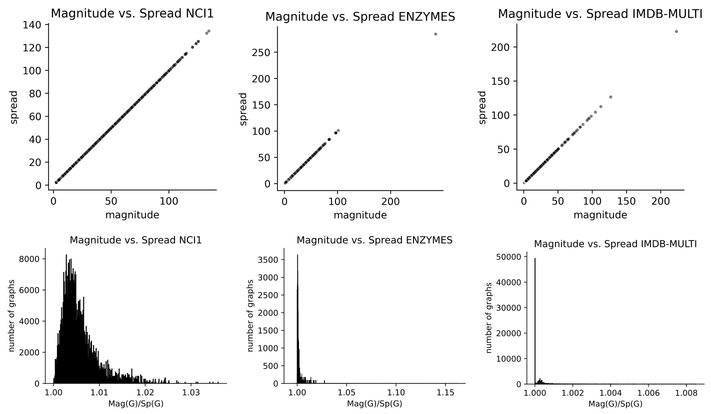


Fig. 5: Comparison between magnitude and spread computed from diffusion distances for all graphs in three datasets, NCI1, ENZYMES and IMDB-Multi.

DiffPool, and MinCutPool). Interpreting the experimental results in Table 1 it is thus of interest that the sizes of the pooled graphs can vary across pooling layers, which might explain some of the difference in performance between the fixed-size methods DiffPool and MinCUT compared to more adaptive pooling methods.

C Extended Results

C.1 Correlation between Magnitude and Spread

As stated in Section 2.2, the magnitude and spread of a metric space are closely related with magnitude giving an upper bound for spread when computed from the same positive definite metric space. Across our experiments on real graph datasets, we further find that this bound in practice can be very tight and magnitude and spread measure very related notions of effective size. More specifically, when computing both magnitude and spread from the diffusion distances detailed in Section 2.3, we observe that magnitude and spread almost coincide for all graphs from the NCI1, ENZYME or IMDB-Multi datasets as illustrated in Figure 5. In fact, magnitude and spread correlate almost perfectly across these three graph datasets (Pearson correlation $r^2 \geq 0.99$). Further, we confirm that across these examples, magnitude is generally greater or equal to spread by a relatively low multiplicative factor close to 1. We therefore find empirical evidence for the fact that spread offers a valid and highly related alternative to magnitude in practice supporting our theoretical analysis of the relationship between spread and magnitude during pooling (Appendix A.3) as well as our observations on the similar performance of MagEdgePool and SpreadEdgePool.

C.2 Evaluating Computational Efficiency

As detailed in Appendix A.2, the computational costs of our algorithm are determined by the costs of computing the edge scores used for pooling. To illustrate how this theoretical discussion translates into practice, we now investigate computational costs empirically in comparison to alternative pooling methods considered throughout this study.

Training Costs We first compare the runtime (in seconds) and memory usage (in MB per cross-validation run) of our pooling methods (MagEdgePool and SpreadEdgePool) to trainable pooling methods in Table 3. Specifically, we train the GNN architecture specified in Appendix B.5 and Section 4.1 using GIN layers across 200 epochs using 10-fold stratified cross-validation. We compare our edge pooling methods (MagEdgePool, SpreadEdgePool) to trainable pooling methods from `torch_geometric`[10] [21, 22] (EdgePool, TopKPool, SAGPool) or from `torch-geometric-pool`[11] (DiffPool, MinCutPool). We record the mean and standard deviation of the runtimes in seconds in Table 3 and GPU memory usage in MB per cross-validation run in Table 4. We note, in particular, that our proposed edge pooling methods, MagEdgePool and SpreadEdgePool, allow for significantly more efficient GNN training than EdgePool as highlighted in Table 3. Similarly, our methods generally improve on the runtimes for dense pooling methods such as DiffPool and MinCutPool. To generalise this runtime comparison to other datasets sizes and experimental setups, we note that our algorithm will scale with dataset size as described in Appendix A.2.

Pre-training Costs Having observed that pre-computed edge-pooling speeds up GNN training times, we further investigate the computational costs of this preprocessing step by comparing the runtime and memory costs of our methods against other non-trainable pooling operators (NDP, NMF and Graclus) prior to training. Table 5 and Table 6 report the computational costs of computing the pooling assignment for all graphs in the datasets. We observe that SpreadEdgePool is generally more efficient than MagEdgePool. Beyond exact computations of our pooling methods (described in Appendix B.4), Table 5 and Table 6 also present approximate versions that reduce the cost of distance computations. Specifically, these approximate versions, referred to as *MagEdgePool** and *SpreaEdgePool**, use the minimum distance to the original nodes to update the (diffusion) distances during edge contraction. This leads to a considerable improvement in runtime and memory usage during pre-training. The results reported in this section highlight one of the limitations of our edge pooling approach, namely, it scales in the number of edges as well as in the number of nodes. Nevertheless, our proposed pooling methods still outperform EdgePool in terms of computational efficiency when considering both pre-processing and training costs.

[10] https://pytorch-geometric.readthedocs.io/en/latest/ available under an MIT license.

[11] https://github.com/tgp-team/torch-geometric-pool available under an MIT license.

Table 3: Training times in seconds compared across pooling methods. The fastest methods are marked in bold.

Method	DHFR	ENZYMES	NCI109	Mutagenicity	IMDB-BINARY	IMDB-MULTI
MagEdge	39.8 ± 6.0	**22.0 ± 0.2**	186.0 ± 9.1	**180.3 ± 29.0**	75.4 ± 1.4	**89.6 ± 6.1**
SpreadEdge	**34.8 ± 1.0**	23.0 ± 0.7	**181.3 ± 29.4**	206.7 ± 20.6	**62.7 ± 0.9**	99.7 ± 4.3
EdgePool	184.1 ± 4.0	209.3 ± 4.1	807.4 ± 22.2	790.7 ± 22.1	362.3 ± 3.4	392.7 ± 11.9
TopKPool	50.7 ± 0.8	24.1 ± 0.6	226.1 ± 21.3	243.8 ± 15.4	88.2 ± 1.1	110.9 ± 2.2
SAGPool	54.4 ± 1.3	25.7 ± 0.4	250.4 ± 25.7	253.0 ± 20.0	92.6 ± 2.1	131.1 ± 1.3
DiffPool	52.2 ± 6.8	36.4 ± 3.3	242.4 ± 12.5	274.0 ± 21.5	106.7 ± 5.5	**74.9 ± 2.0**
MinCut	**33.8 ± 2.5**	28.2 ± 1.7	210.3 ± 34.4	201.0 ± 3.7	93.2 ± 1.6	132.8 ± 3.2

Table 4: Memory usage in MB compared across pooling methods. The most efficient methods are marked in bold.

Method	DHFR	ENZYMES	NCI109	Mutagenicity	IMDB-BINARY	IMDB-MULTI
MagEdge	**84.0 ± 0.1**	**91.2 ± 2.5**	**97.0 ± 1.1**	**94.8 ± 1.0**	**283.4 ± 28.0**	**197.6 ± 17.8**
SpreadEdge	**84.0 ± 0.1**	**90.6 ± 2.3**	**96.8 ± 1.0**	**95.2 ± 1.0**	**283.4 ± 28.0**	**197.4 ± 17.8**
EdgePool	125.6 ± 10.4	148.4 ± 15.0	118.6 ± 7.7	118.0 ± 7.4	666.6 ± 106.6	526.2 ± 89.2
TopKPool	89.6 ± 9.0	94.4 ± 4.0	105.0 ± 1.4	105.8 ± 6.5	**259.4 ± 31.5**	**193.8 ± 17.5**
SAGPool	104.2 ± 0.6	94.2 ± 3.8	105.2 ± 1.4	107.8 ± 8.1	**273.4 ± 30.1**	**209.0 ± 17.9**
DiffPool	90.2 ± 10.2	94.6 ± 2.5	103.8 ± 1.1	299.8 ± 31.9	**258.0 ± 40.3**	240.8 ± 26.4
MinCut	90.6 ± 10.3	94.2 ± 2.4	103.6 ± 1.0	288.4 ± 38.3	**258.0 ± 40.3**	**196.6 ± 26.0**

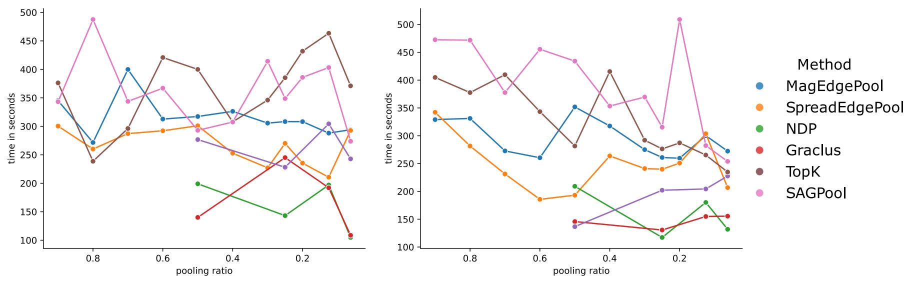
Table 5: Pre-training times in seconds compared across non-trainable pooling methods. The fastest method is marked in bold. The fastest approximation of our pooling method is marked in cursive.

Method	DHFR	ENZYMES	Mutagenicity	NCI109	IMDB-BINARY	IMDB-MULTI
MagEdge	61.9	197.4	4765.1	678.8	801.8	480.2
MagEdge*	24.6	21.0	81.3	74.9	77.4	69.9
SpreadEdge	66.7	68.3	316.1	208.8	287.2	243.8
SpreadEdge*	*10.9*	*16.0*	*52.3*	*55.6*	*56.0*	*66.0*
NDP	5.6	4.1	**37.1**	32.6	5.8	7.0
NMF	7.8	10.8	39.0	32.1	8.7	9.0
Graclus	**3.8**	**2.9**	54.5	**18.2**	**4.5**	**6.0**

Table 6: Pre-training memory usage in MB compared across non-trainable pooling methods. The most efficient method is marked in bold. The most efficient approximation of our pooling method is marked in cursive.

Method	DHFR	ENZYMES	Mutagenicity	NCI109	IMDB-BINARY	IMDB-MULTI
MagEdgePool	76.2	163.8	158.9	179.0	162.1	93.1
MagEdgePool*	83.7	*34.8*	*47.0*	*72.2*	*11.5*	65.2
SpreadEdgePool	61.9	60.1	177.9	93.1	204.1	75.8
SpreadEdgePool*	*33.6*	54.1	60.5	72.9	53.2	*23.8*
NDP	**3.0**	**5.0**	**7.0**	13.4	5.1	6.5
NMF	3.8	5.9	39.9	**10.5**	4.2	5.9
Graclus	15.8	7.5	17.5	59.5	**1.0**	**1.5**

Fig. 6: Runtime comparison for training the GNNs reported in Section 4.1 for NCI1 using different pooling layers. Plots show the mean time in seconds per run using general convolutional layers (left) or GIN layers (right).



Furthermore, the pre-training costs of our method, range in a number of seconds, need to be computed only once per dataset and the memory requirements remain below what is required by GNN training. Hence, while the scalability to very large graphs is a limitation, we find that our proposed pooling methods scale sufficiently well to standard graph datasets. In practice, based on the computational complexity (Appendix A.2), we recommend that our pooling method is particularly suitable for small to medium graphs that show a certain degree of sparsity rather than being fully connected. For future work, we believe that there is a strong potential for adapting our methods to scale on large graphs. For instance, edge score calculations could be parallised, sampling heuristics could restrict the edge score computations to a subset of candidate edges, or edge scores could be estimated from local subgraphs to improve the computations.

Runtimes across Pooling Ratios Further, we compare the runtimes of training the models used in Section 4.3 to compare the accuracy of different pooling methods across varying pooling ratios. Figure 6 then reports the mean runtime of training the GNN on one CV-fold for different choices of pooling ratios and pooling methods for the NCI1 dataset using either general convolutional layers or GIN layers. All models are trained as specified in Section 4.1 on a single GPU with 32GB memory. Notably, we observe that MagEdgePool and SpreadEdgePool overall perform on par with alternative pooling methods in terms of runtimes. SpreadEdgePool has a consistent advantage over MagEdgePool due to the higher computational efficiency of computing spread rather than magnitude. Notice that for increasing pooling ratios, our algorithm re-computes the edge scores repeatedly, leading to a less pronounced decrease in computational costs than alternative methods. Nevertheless, we conclude that it is generally more efficient to apply SpreadEdgePool than to rely on trainable approaches, such as TopK and SAGPool for this dataset, indicating the computational benefit of non-trainable graph pooling operations.

Table 7: Mean and standard deviation of the reconstruction MSE for reconstructing the original node positions from the pooled graph representations for different example graphs and pooling methods. Our proposed algorithm, SpreradEdgePool, does well at faithfully encoding the feature representations.

	Ring	**Sensor**	**barbell**	**community**	**erdosrenyi**	**torus**
SpreadEdge	5.47e-07 ± 2.63e-07	2.78e-05 ± 3.04e-07	3.42e-04 ± 1.42e-06	3.71e-03 ± 5.80e-05	6.49e-07 ± 3.33e-07	5.29e-07 ± 1.33e-07
NDP	3.08e-07 ± 3.57e-07	4.07e-05 ± 4.57e-06	4.54e-04 ± 2.01e-05	2.52e-01 ± 9.50e-06	1.46e-06 ± 1.18e-06	5.68e-07 ± 1.02e-07
Graclus	6.87e-04 ± 7.56e-07	2.67e-06 ± 2.31e-06	1.82e-03 ± 3.22e-07	2.42e+00 ± 2.11e-04	4.76e-02 ± 3.75e-07	7.10e-07 ± 8.60e-08
NMF	4.78e-07 ± 2.95e-07	1.96e-05 ± 1.39e-05	5.80e-04 ± 4.53e-07	6.06e-01 ± 1.43e-04	5.04e-07 ± 3.31e-07	2.52e-07 ± 2.93e-07
TopK	1.21e-01 ± 8.23e-03	5.83e-03 ± 2.16e-03	1.55e-02 ± 1.10e-02	6.03e+00 ± 2.21e+00	5.30e-03 ± 7.49e-03	1.72e-01 ± 8.51e-03
SAGPool	1.45e-01 ± 2.52e-02	2.01e-03 ± 2.73e-03	4.12e-02 ± 4.18e-02	4.76e+00 ± 1.57e+00	9.60e-05 ± 1.35e-04	1.88e-01 ± 4.63e-02
DiffPool	8.63e-06 ± 4.73e-06	3.50e-04 ± 8.32e-05	6.50e-04 ± 1.01e-06	2.14e-01 ± 2.84e-01	3.74e-04 ± 1.46e-04	5.17e-05 ± 8.90e-06
MinCut	2.55e-06 ± 2.68e-06	6.56e-06 ± 3.86e-06	2.35e-06 ± 1.50e-06	1.80e-04 ± 2.01e-04	1.44e-06 ± 5.24e-07	1.49e-06 ± 9.81e-07

C.3 Node Feature Preservation and Expressivity

Graph pooling should not only preserve graph structure, but also preserve relevant node feature information during pooling. That is, pooling is frequently used after initial rounds of message passing and data representations learnt by previous layers should be respected and effectively encoded by the pooling procedure [26]. One way of investigating node feature retention during pooling, is to evaluate how well a graph can be reconstructed from its pooled version. We follow the experimental setup by [26] to investigate. In particular, this experiment uses a model architecture, similar to the model proposed in Section 4.1, where each graph gets pooled to around 50% of nodes after the initial MLP and GNN layer. Then, the pooled graphs are up-scaled again by reversing the node selection step used by each pooling layer. From these unpooled graph representations, a further GNN and post-processing MLP layer are trained and the task is set to output the reconstructed node feature representation. This model is trained on each example graph using Adam to minimize the mean squared error (MSE) between the input and output node features with a learning rate of 0.0005 and early stopping on the training loss with a patience of 1000 epochs and a tolerance of 10^{-6}. Each experiment is repeated three times across different random seeds. See [26] for further explanations on the model architecture and experimental setup.

For this experiment, we expect SpreadEdgePool to perform comparably well as we specifically designed our pooling algorithm so that features are averaged during pooling. Further restricting the number of times a node can be merged effectively prevents the collapse of entire portions of the graph, which aids reconstruction. SpreadEdgePool pooling thus successfully encodes node information while allowing for a flexible choice of pooling ratio. This is confirmed by the results in Table 7, which highlight that SpreadEdgePool overall performs well at the features reconstruction task, especially for the sensor graph reaching low reconstruction errors. Alternative methods, in particular node drop approaches such as TopK and SAGPool, show notably worse feature preservation during this experiment indicating the benefits of more expressive pooling operations, such as SpreadEdgePool. Note that the results in Table 7 capture one specific aspect of feature preservation, namely how well the features of specific example graphs can

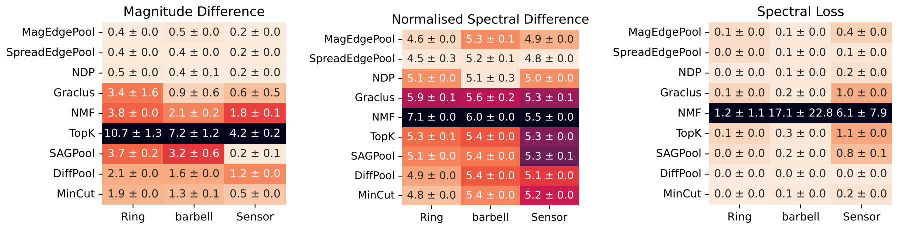


Fig. 7: Structure preservation measures for the examples in Figure 1. Pooling is repeated for three different random seeds and the annotations report the means and standard deviations of the structure preservation scores.

be reconstructed. This experiment does not assess the generalisation capability of pooling layers. Further, the experimental setup assumes that it is relevant to preserve all node features during pooling, which might not be realistic in practice, where the aim of pooling could be to solely encode task-relevant feature representations. Nevertheless, as discussed above, this extended experiment gives evidence to support that our proposed pooling algorithm, SpreadEdgePool, is capable of outputting expressive feature representations and aggregates node features in a faithful manner, which is likely one of the reasons for its high performance in graph classification tasks.

C.4 Overview Experiment

Expanding on the qualitative comparison between the example graphs in Figure 1, Figure 7 shows quantitative structure preservation measures for all example graphs and pooling methods. Specifically, we summarise the spectral loss [26], the spectral distance between the normalised graph Laplacians, and the magnitude difference between the pooled and original graphs as further detailed in Section 4.2. Figure 7 demonstrates that SpreadEdgePool and MagEdgePool do not only reach low magnitude differences across these three example graphs, they also show comparatively low spectral distances supporting our findings in Section 4.2. Further, methods that show worse visual preservation of graph structures, such as NMF, TopK, DiffPool, and MinCut, also reach consistently higher magnitude differences and spectral distances supporting our claim that these methods fail to faithfully preserve graph structures during pooling to varying extents.

C.5 Graph Classification

Table 8 further reports extended classification results on additional datasets extending on the results shown in Table 1. We chose not to report on these datasets in the main text because they showed fewer and less notable differences between pooling methods. Note that for the open graph benchmark MolHIV dataset, instead of using stratified cross-validation, we evaluate each model across predefined training, test and validation splits and evaluate their performance across 5

Table 8: Classification performance of different pooling layers. For each dataset, the best performing model is marked in bold and models that do not perform significantly different from the best performing model are coloured green.

Method	**MolHIV (AUROC)**	**MUTAG**	**COX2**	**BZR**	**BZR_MD**	**COX2_MD**	**DHFR_MD**	**ER_MD**	**AIDS**
No Pooling	74.4 ± 0.7	81.6 ± 6.3	83.8 ± 3.8	76.1 ± 9.1	73.7 ± 5.3	70.2 ± 8.1	71.3 ± 1.9	74.9 ± 0.8	99.0 ± 0.1
MagEdge	64.6 ± 8.7	84.0 ± 7.4	86.0 ± 7.3	**88.1 ± 10.0**	65.9 ± 2.2	76.5 ± 9.2	**77.9 ± 9.4**	79.9 ± 7.1	**99.7 ± 0.1**
SpreadEdge	70.1 ± 2.9	85.9 ± 7.4	85.1 ± 5.6	87.4 ± 9.5	69.7 ± 10.0	**77.1 ± 9.0**	74.6 ± 11.1	83.1 ± 4.0	**99.7 ± 0.1**
NDP	65.2 ± 7.3	**91.6 ± 2.4**	**86.1 ± 7.1**	86.3 ± 9.4	72.3 ± 11.3	73.9 ± 10.6	72.2 ± 11.4	**84.2 ± 1.7**	99.6 ± 0.1
Graclus	70.1 ± 5.3	87.4 ± 9.5	79.2 ± 12.5	80.1 ± 7.1	72.0 ± 9.6	75.4 ± 9.9	71.1 ± 12.3	81.7 ± 3.9	99.5 ± 0.1
NMF	73.2 ± 1.3	84.0 ± 9.4	83.8 ± 8.7	80.8 ± 8.5	69.0 ± 10.2	70.4 ± 6.6	70.8 ± 8.6	80.9 ± 2.0	97.0 ± 1.7
TopK	72.7 ± 1.8	81.9 ± 3.9	76.4 ± 7.2	75.8 ± 8.0	68.4 ± 8.5	65.9 ± 7.0	69.5 ± 3.3	74.0 ± 1.0	99.3 ± 0.1
SAGPool	**74.3 ± 3.2**	82.9 ± 2.3	76.8 ± 7.8	77.8 ± 5.0	66.8 ± 7.0	67.0 ± 6.4	70.5 ± 2.6	75.6 ± 1.2	99.0 ± 0.1
DiffPool	72.1 ± 1.0	83.3 ± 2.8	76.9 ± 6.5	76.8 ± 10.1	72.3 ± 3.0	69.2 ± 2.1	67.9 ± 2.6	73.4 ± 1.1	98.8 ± 0.2
MinCut	70.3 ± 3.0	80.6 ± 3.7	79.1 ± 4.4	70.8 ± 8.6	69.0 ± 5.0	70.0 ± 3.7	69.7 ± 1.9	73.3 ± 1.6	99.2 ± 0.1

random seeds. Further, we report AUROC as the performance metric for MolHIV because is the suggested evaluation metric for this very imbalanced dataset. All other results are reported as in Table 1 via the mean and standard deviation of the test accuracy across 10-fold stratified cross-validation. In agreement with our main results, we observe that MagEdgePool and SpreadEdgePool constitute high performing general-purpose pooling methods that reach top performance across these extended datasets. In particular, for smaller biological datasets, such as MUTAG, BZR, or COX2, pooling via magnitude or spread notably improves on the GNN that uses no pooling layer, which indicates the beneficial effects of structure-aware pooling for graph learning.

C.6 Comparison with EdgePool

To extend our evaluation, we compare with the `pytorch_geometric` implementation of EdgePool [15, 16]. To do so, we implement the GNN architecture specified in Section 4.1 in `PyTorch` and repeat our main experiment reported in Table 1. We find that there is no significant difference in performance between our methods and EdgePool when computed on e.g. IMDB-M, where EdgePool reaches an accuracy of 47.6 ± 3.3, or on IMDB-B, where EdgePool reaches an accuracy of 70.9 ± 4.5. We expect this same pattern to hold across further datasets.

As a major advantage, our methods have notably lower computational costs during training than EdgePool as reported in Appendix C.2. Our methods thus make edge-contraction pooling scalable to larger datasets and enable significantly faster GNN training. On IMDB-B for example, EdgePool takes 362.2 seconds and 666.6 MB for 200 epochs during training, but our method SpreadEdgePool only requires 62.7 seconds and 283.4 MB. We thus find that learning feature-based edge scores as done by EdgePool, is not necessary to ensure classification performance, but rather adds a notable computational burden. The main advantage of our pooling methods over EdgePool is thus improved scalability as well as the flexible choice of pooling ratio.

C.7 Preserving Graph Structure

Extending on the results reported in Figure 3 and Section 4.2, we further report the distribution of structure preservation measures across pooling ratios. Specifically, Figure 8 shows line plots that summarise the mean magnitude difference relative to the original graph and the normalised spectral distance between all original and pooled graphs from the NCI1 dataset in the leftmost column. The remaining plots illustrate the quantiles of the same measures split up per pooling method. Overall, these individual plots support our assessment that MagEdgePool and SpreadEdgePool consistently reach low magnitude differences and comparably low spectral distances, with these trends being more pronounced in terms of the relative difference in magnitude after pooling.

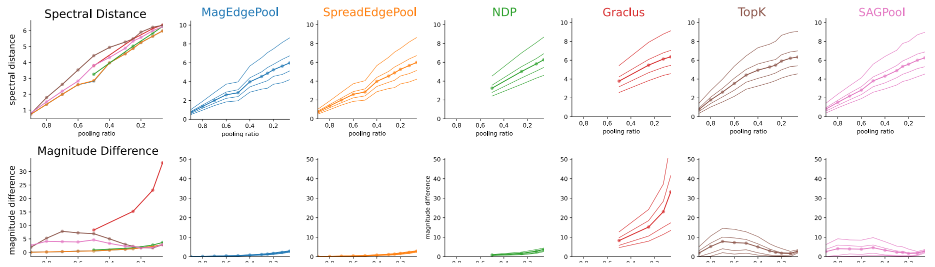


Fig. 8: Structure preservation for the NCI1 dataset across varying pooling ratios. Top row: The spectral distance between the normalised Laplacians of the original and pooled graphs. Bottom row: The relative difference in magnitude. Bold lines show the mean values of each score across graphs and thin lines the 10%, 25%, 75% and 90% quantiles.