Sheaf-based Positional Encodings for Graph Neural Networks

Yu He[*] HEYU@CS.STANFORD.EDU
Stanford University
Cristian Bodnar CBODNAR@MICROSOFT.COM
Microsoft Research AI4Science
Pietro Liò PL219@CAM.AC.UK
University of Cambridge

Editor: Sophia Sanborn, Christian Shewmake, Simone Azeglio, Nina Miolane

Abstract

Graph Neural Networks (GNNs) work directly with graph-structured data, capitalising on relational information among entities. One limitation of GNNs is their reliance on local interactions among connected nodes. GNNs may generate identical node embeddings for similar local neighbourhoods and fail to distinguish structurally distinct graphs. Positional encodings help to break the locality constraint by informing the nodes of their global positions in the graph. Furthermore, they are required by Graph Transformers to encode structural information. However, existing positional encodings based on the graph Laplacian only encode structural information and are typically fixed. To address these limitations, we propose a novel approach to design positional encodings using sheaf theory. The sheaf Laplacian can be learnt from node data, allowing it to encode both the structure and semantic information. We present two methodologies for creating sheaf-based positional encodings, showcasing their efficacy in node and graph tasks. Our work advances the integration of sheaves in graph learning, paving the way for innovative GNN techniques that draw inspiration from geometry and topology.

1. Introduction

Real-world datasets often possess inherent graph structures, with data entities connected through intrinsic relationships. Graph Neural Networks (GNNs) (Kipf and Welling, 2016; Gilmer et al., 2017; Veličković et al., 2018) have proven highly effective in leveraging this relational information, showcasing their effectiveness in diverse applications. These applications span fields such as protein interface prediction (Fout et al., 2017), social networks (Fan et al., 2019), and knowledge graphs (Xu et al., 2019).

However, GNNs are limited in expressive power (Xu et al., 2018). GNNs generally follow a message-passing paradigm (Gilmer et al., 2017; Veličković, 2022), where each node aggregates messages from its neighbours. While this approach benefits from scalability, it also confines each node's knowledge to its immediate local surroundings. This limited perspective presents a challenge for GNNs in distinguishing between nodes that share similar local neighbourhoods. Additionally, when making predictions at the graph level, typically through aggregating all node embeddings, GNNs can struggle to differentiate between structurally distinct graphs.

[*] Work done while the author was at the University of Cambridge.

To address this issue, positional encodings (PEs) offer a solution by providing nodes with awareness of their global positions in the graph. PEs have previously found applications in natural language processing, where they encode token positions in sequences for Transformers (Vaswani et al., 2017). In the context of GNNs, we similarly employ PEs to encode node positions within a graph. However, defining PEs for graph nodes poses a unique challenge, as nodes in a graph lack a natural order for sequential indexing. Furthermore, PEs play a crucial role in Graph Transformers (Kreuzer et al., 2021; Dwivedi and Bresson, 2020; Ying et al., 2021; Mialon et al., 2021; Rampášek et al., 2023; Müller et al., 2023). Graph Transformers apply attention mechanisms across a fully connected graph, necessitating PEs to compensate for the loss of structural information from the input graph.

The graph Laplacian has emerged as a prominent choice for crafting efficient graph PEs (Dwivedi et al., 2020; Kreuzer et al., 2021). The eigendecomposition of the graph Laplacian yields an embedding space that respects the underlying graph topology (Belkin and Niyogi, 2003). This space can be precomputed and integrated as additional input. However, challenges surface when dealing with intricate node relationships. For instance, while homophilic graphs exhibit nodes with similar representations positioned closer together ('like attracts like'), heterophilic graphs have dissimilar nodes connected. The graph Laplacian solely encodes adjacency information and remains oblivious to relationships within node data. Consequently, the PE space constructed from the graph Laplacian may prove inadequate for such complex graphs.

Our proposed solution involves leveraging the sheaf Laplacian to mitigate this limitation. A cellular sheaf, a mathematical construct rooted in geometry and topology, possesses the capability to encode algebraic and topological structures, parameterised by the node data. This attribute empowers the sheaf Laplacian to generate a positional embedding space that encompasses both the structural and semantic facets of the graph. Therefore, it can be more suitable for more intricate domains, including scenarios involving heterophilic graphs, where complex node relationships are prevalent.

2. Background

2.1. Graph Neural Networks

Graph Neural Networks (GNNs) (Kipf and Welling, 2016; Gilmer et al., 2017; Veličković et al., 2018) are designed to operate directly on graph-structured data, capitalising on the relational information among nodes. In a given graph $G = (V, E)$, with V representing the node set and E the edge set, each node is associated with a feature vector $\mathbf{h}_v \in \mathbb{R}^{d_h}$, where d_h denotes the dimension. GNNs generally follow a message-passing paradigm (Gilmer et al., 2017; Veličković, 2022), where nodes exchange messages with their connected neighbours to update their representations $\mathbf{h}_v$ via:

$$\mathbf{h}_v^{l+1} = \phi^l\Big(\mathbf{h}_v^l, \bigoplus_{u \in \mathcal{N}(v)} \psi^l(\mathbf{h}_v^l, \mathbf{h}_u^l)\Big) \tag{1}$$

where l is the layer number, $\mathcal{N}(v) = \{u \in V | (u, v) \in E\}$ is the one-hop neighbourhood of node v, and ϕ^l, ψ^l are commonly implemented as MLPs. At each layer l, a node computes a message from its connected neighbours using a message function ψ^l, then aggregates the

message with a permutation-invariant aggregation operator $\bigoplus$. Finally, the node uses the message to update its node representation with a readout function ϕ^l.

The message-passing paradigm, which centres on pairwise communication within local neighbourhoods, provides scalability. However, this locality also introduces a challenge in expressivity. GNNs are proven to be at most as powerful as the 1-WL (Weisfeiler-Lehman) test (Xu et al., 2018; Morris et al., 2018), a heuristic for graph isomorphism test based on passing random hashes of node colours along the edges. In fact, GNNs may produce identical embeddings for nonisomorphic graphs, such as regular graphs (Cai et al., 1989; Douglas, 2011; Evdokimov and Ponomarenko, 1999), and may even fail to recognise simple substructures. This limitation poses a significant challenge in real-world scenarios, such as molecules, where nonisomorphic graphs often carry different labels.

Positional Encodings To address the challenge posed by the locality constraint, we can employ positional encodings (PEs) (You et al., 2019; Dwivedi et al., 2020; Li et al., 2020; Dwivedi and Bresson, 2020). PEs serve the purpose of explicitly informing nodes about their global positions within the graph. This global awareness, facilitated by PEs, empowers GNNs to differentiate between nonisomorphic graphs and identify various substructures effectively. Typically, PEs are concatenated with node or edge features and introduced as inputs to GNNs. One effective method for computing PEs in graphs involves utilising the graph Laplacian (Dwivedi et al., 2020; Kreuzer et al., 2021; Lim et al., 2022). Its eigendecomposition is a spectral technique that embeds the graph into an Euclidean space, deriving a unique PE for each node.

Graph Transformers Graph Transformers (GTs) (Kreuzer et al., 2021; Dwivedi and Bresson, 2020; Ying et al., 2021; Mialon et al., 2021; Rampášek et al., 2023; Müller et al., 2023) generalise the Transformer architecture (Vaswani et al., 2017) to graphs. GTs apply attention over a fully-connected graph to capture global and long-range interactions, addressing the over-smoothing (Li et al., 2018) and over-squashing (Alon and Yahav, 2020) issues from sparse graphs. However, not using the input graph also means that Graph Transformers lose the structural information, only retaining the number of nodes. Therefore, the expressive power of using a fully-connected graph is equivalent to DeepSets, (Zaheer et al., 2018) where no edge presents. To restore the topological awareness on the input graph, it is critical for GTs to equip with PEs to improve their expressivity while maintaining the advantages from the Transformer layers.

2.2. Sheaf Theory

In algebraic topology, a cellular sheaf (Curry, 2014; Singer and Wu, 2011) is a mathematical object associated with a graph by attaching vector spaces to nodes and edges as shown in Figure 1. It also defines a map between these vector spaces for each incident node-edge pair, specifying the consistency constraints of the data.

Definition 1 (Cellular sheaf) *A cellular sheaf $(G, \mathcal{F})$ on an undirected graph $G = (V, E)$ consists of:*

- *A vector space $\mathcal{F}(v)$ for each vertex $v \in V$.*
- *A vector space $\mathcal{F}(e)$ for each edge $e \in E$.*

- *A linear map* $\mathcal{F}_{v \trianglelefteq e} : \mathcal{F}(v) \to \mathcal{F}(e)$ *for each incident node-edge pair* $v \trianglelefteq e$.

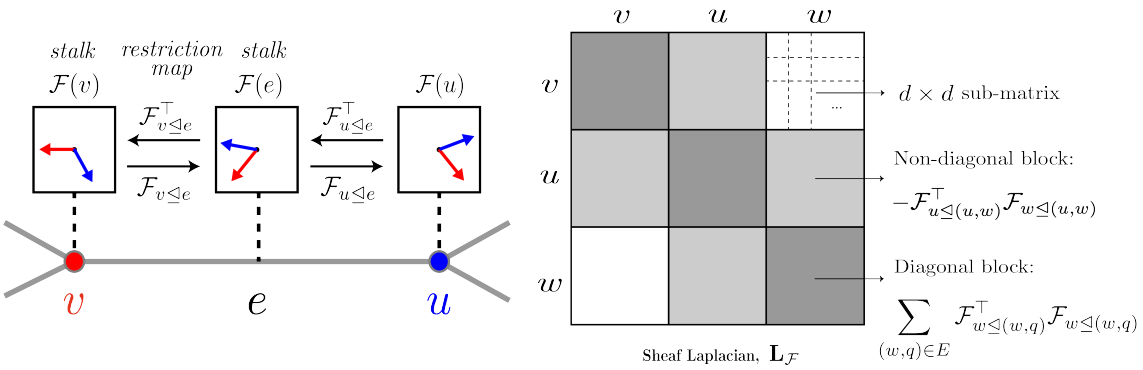

Figure 1: (a) A sheaf $(G, \mathcal{F})$ for an edge e connecting two nodes v and u. (b) The sheaf Laplacian. Figures adapted from Bodnar et al. (2022).

We call the vector spaces of the nodes and edges as *stalks*, and the linear maps as *restriction maps*. The space of *0-cochains* $C^0(G; \mathcal{F}) := \bigoplus_{v \in V} \mathcal{F}(v)$ is the space formed by all the stalks associated with the nodes of the graph, where $\bigoplus$ denotes the direct sum of vector spaces. The *sheaf Laplacian* operator for a given cellular sheaf measures the aggregated "disagreement of opinions" at each node.

Definition 2 (Sheaf Laplacian) *Given a cellular sheaf* $(G; \mathcal{F})$*, the sheaf Laplacian is a linear map* $\mathbf{L}_{\mathcal{F}} : C^0(G, \mathcal{F}) \to C^0(G, \mathcal{F})$*, which can be defined node-wise as* $\mathbf{L}_{\mathcal{F}}(\mathbf{x})_v = \sum_{v, u \trianglelefteq e} (\mathcal{F}_{v \trianglelefteq e} \mathbf{x}_v - \mathcal{F}_{u \trianglelefteq e} \mathbf{x}_u)$*. Here,* $\mathbf{x} \in C^0(G; \mathcal{F})$ *is a 0-cochain, and* $\mathbf{x}_v$ *is the vector in* $\mathcal{F}(v)$ *of node* v*.*

Given a graph G with n nodes, assuming all stalks have a fixed dimension d, the sheaf Laplacian is an $nd \times nd$ positive semi-definite block matrix. As illustrated in Figure 1, the sheaf Laplacian has diagonal blocks $\mathbf{L}_{\mathcal{F}_{vv}} = \sum_{v \trianglelefteq e} \mathcal{F}^{\top}_{v \trianglelefteq e} \mathcal{F}_{v \trianglelefteq e}$, and non-diagonal blocks $\mathbf{L}_{\mathcal{F}_{vu}} = -\mathcal{F}^{\top}_{v \trianglelefteq e} \mathcal{F}_{u \trianglelefteq e}$. The normalised sheaf Laplacian is $\Delta_{\mathcal{F}} := \mathbf{D}^{-1/2} \mathbf{L}_{\mathcal{F}} \mathbf{D}^{-1/2}$, where $\mathbf{D}$ is the block-diagonal of $\mathbf{L}_{\mathcal{F}}$.

Sheaf Laplacian generalises graph Laplacian The graph Laplacian is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix of the graph. The graph Laplacian is a trivial sheaf, by setting all the stalks to scalars ($d = 1$, where d is the stalk dimension) and the restriction maps to identity functions. In contrast, the sheaf Laplacian extends the capabilities of the graph Laplacian by allowing for a higher dimension ($d \geq 1$). This extension empowers the sheaf to encode richer information, including the semantic knowledge embedded by the node data. GNNs have been extended to operate over a sheaf structure in prior works (Hansen and Gebhart, 2020; Bodnar et al., 2022; Barbero et al., 2022), offering solutions to challenges like heterophily and oversmoothing often encountered in conventional GNNs.

3. Sheaf-based Positional Encodings

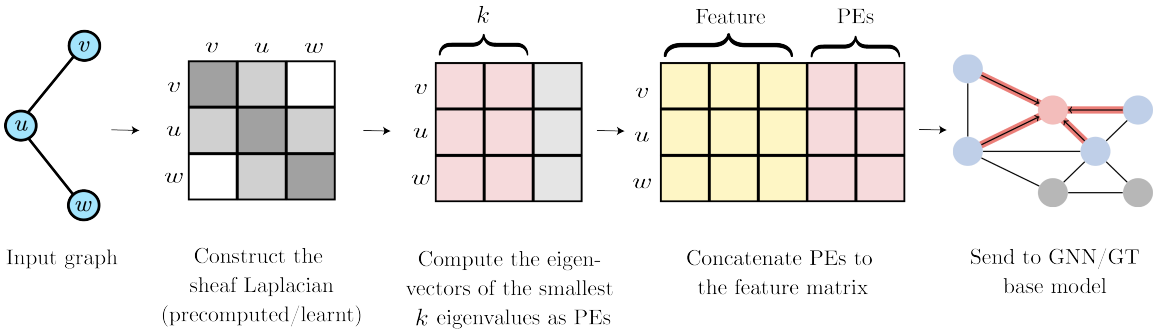

Figure 2: Framework of sheaf-based PEs, where the sheaf can be constructed via precomputed and learnt methods.

3.1. Precomputed sheaf Laplacian (ConnLap)

PEs that rely on the graph Laplacian are typically computed during the pre-processing phase using the adjacency matrix of the graph. Following a similar approach, we suggest constructing PEs using the sheaf Laplacian through a precomputed method. When we enforce the restriction maps to take the form of an orthogonal matrix, we arrive at a specialised variant of the sheaf Laplacian, known as the connection Laplacian (Singer and Wu, 2011). The connection Laplacian can be thought of as a discretised representation of the vector bundle, which draws an analogy to the concept of parallel transport on a manifold.

The concept of "transport" for a sheaf can be denoted as $\mathbf{P}^{\gamma}_{v\to u} : \mathcal{F}(v) \to \mathcal{F}(u)$. This transport operation involves composing restriction maps along the edges, facilitating the flow of information from the stalk $\mathcal{F}(v)$ to the stalk $\mathcal{F}(u)$:

$$\mathbf{P}^{\gamma}_{v\to u} = (\mathcal{F}^{\top}_{u\trianglelefteq e_l}\mathcal{F}_{v_l\trianglelefteq e_l})...(\mathcal{F}^{\top}_{v_1\trianglelefteq e_1}\mathcal{F}_{v\trianglelefteq e_1}) \tag{2}$$

where $v, u \in V$ are two nodes in the graph, and $\gamma_{v\to u} = (v, v_1, ..., v_l, u)$ denotes the sequence of nodes on the path from v to u. We can then compute a connection Laplacian by an analogy to the parallel transport on a manifold. Given a manifold $\mathcal{M}$, each point in the manifold $\mathbf{x} \in \mathcal{M}$ is associated with a tangent space $\mathcal{T}_{\mathbf{x}}\mathcal{M}$. For two nodes v and u, its transport maps $\mathbf{P}^{\gamma}_{v\to u}$ from $\mathcal{F}(v)$ to $\mathcal{F}(u)$ correspond to the parallel transport from $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ to $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$, where $\mathbf{x}_v$ and $\mathbf{x}_u$ are the associated node data. Through such an analogy, when v and u are connected by an edge e in the graph, there is a canonical path to perform the transport on the manifold. Therefore, we can derive from Equation 2 for the transport as $\mathbf{P} = \mathbf{P}^{\gamma}_{v\to u} = \mathcal{F}^{\top}_{u\trianglelefteq e}\mathcal{F}_{v\trianglelefteq e}$.

The last question is how to compute the parallel transport from $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ to $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$ on the manifold $\mathcal{M}$ in the context of graph data. In Barbero et al. (2022), they propose a method to precompute this transport for Sheaf Neural Networks (Hansen and Gebhart, 2020). Here, we explore its application to construct sheaf-based PEs. Detailed description of the algorithm can be found in Appendix B. In summary, the method comprises two key

steps. First, we compute the orthonormal bases of $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ and $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$ using local PCA, defined by the 1-hop neighbourhood of the nodes. Secondly, we compute the orthogonal mapping $\mathbf{O}_{vu}$ by optimal aligning the two bases, which approximates the parallel transport operator (Singer and Wu, 2011). It is worth noting that this method relies on the manifold assumption, which posits that although data points exist in a high-dimensional ambient space $\mathbb{R}^p$, they can be effectively embedded onto a lower-dimensional Riemannian manifold $\mathcal{M}^q$, where $q \ll p$. This assumption underpins the successful application of the approach.

Learnable precomputed PEs (LSPE) In some cases, the manifold assumption does not hold, especially when the node features are not sufficiently informative. To tackle the problem, we optionally allow the PEs to evolve after initialising them with the precomputed sheaf. We adopt the framework proposed in Dwivedi et al. (2021) which updates PEs with message-passing. At initialisation, for each node $v \in V$, the precomputed sheaf-based PEs are embedded to the same dimension as the node features: $\mathbf{h}_v^0 = \mathbf{x}_v$ and $\mathbf{p}_v^0 = \text{MLP}(\mathbf{p}_v^{\text{ConnLap}})$. Here, $\mathbf{h}_v$ is the node representation, $\mathbf{x}_v$ is the input node feature, $\mathbf{p}_v$ is the node PE, and $\mathbf{p}_v^{\text{ConnLap}}$ is its initialisation with the connection Laplacian. During training, both node and positional representations are updated at each layer through $\mathbf{h}_v^{l+1} = f_h([\mathbf{h}_v^l||\mathbf{p}_v^l], \{[\mathbf{h}_u^l||\mathbf{p}_u^l]\}_{u\in\mathcal{N}(v)})$ and $\mathbf{p}_v^{l+1} = f_p(\mathbf{p}_v^l, \{\mathbf{p}_u^l\}_{u\in\mathcal{N}(v)})$. Here, l is the layer number, $||$ denotes concatenation, f_h is the update function for node representations depending on the GNN layer used, and f_p swaps the activation function in f_h to `tanh` to accommodate both positive and negative values in positional coordinates.

3.2. Learnt sheaf Laplacian (SheafLap)

We also explore the construction of PEs derived from a learnt sheaf, following the method proposed in Bodnar et al. (2022). In this approach, given a restriction map $\mathcal{F}_{v\trianglelefteq e} : \mathcal{F}(v) \to \mathcal{F}(e)$, where v and e represent an incident node-edge pair, we approximate the restriction map using a learnable parametric function $\Phi : \mathbb{R}^{d\times 2} \to \mathbb{R}^{d\times d}$. That is, $\mathcal{F}_{v\trianglelefteq e:=(v,u)} = \Phi(\mathbf{x}_v, \mathbf{x}_u)$, where $\mathbf{x}_v$ and $\mathbf{x}_u$ are node features for v and u. In practice, this parametric function is implemented as an MLP that takes the concatenation of the two node features as input. The output is then reshaped to match the dimensions of the restriction maps. The formulation can be expressed as $\Phi(\mathbf{x}_v, \mathbf{x}_u) = \sigma(\mathbf{W}[\mathbf{x}_v||\mathbf{x}_u] + \mathbf{b})$, where $||$ denotes matrix concatenation, $\mathbf{W}$ represents the weight matrix, and an optional bias term $\mathbf{b}$ is included. The activation function σ introduces non-linearity. In Bodnar et al. (2022), they show that if Φ possesses adequate capacity and the features are diverse enough, this approach enables the learning of any type of sheaves over a graph. We restrict Φ to learn a sheaf Laplacian with an orthogonal matrix to strike a balance between efficiency and generality. It also corresponds to the connection Laplacian discussed earlier in Section 3.1 with the precomputed method.

3.3. Sheaf-based Positional Encodings

We construct PEs using the sheaf Laplacian with the following steps. Given an $nd \times nd$ sheaf Laplacian matrix $\mathbf{L}$, where n is the number of nodes and d is a chosen stalk dimension, we first perform eigendecomposition $\mathbf{L} = \mathbf{U}^T\mathbf{\Lambda}\mathbf{U}$. Here, $\mathbf{\Lambda}$ denotes the $n \times d$ eigenvalues and $\mathbf{U}$ denotes the $nd \times nd$ eigenvector matrix. Given d_x as the dimension of each node feature,

the node feature matrix becomes $\mathbf{X} \in \mathbb{R}^{n \times d_x}$. We take the eigenvectors corresponding to the smallest k eigenvalues from $\mathbf{U}$, reshape them to $n \times kd$, and concatenate them with $\mathbf{X}$. This results in a new feature matrix $\mathbf{X}' \in \mathbb{R}^{n \times (d_x + kd)}$, which are then fed into the GNN as inputs. We note that the eigendecomposition poses a sign ambiguity problem (Dwivedi et al., 2020), which can be solved by SignNet (Lim et al., 2022). We leave such possible improvements for future exploration.

4. Evaluation

4.1. Node-level tasks

To evaluate the effectiveness of our PEs in assisting GNNs to differentiate nodes and handle the complex relationships between node data, we test our sheaf-based PEs on a set of node-level tasks following previous works (Pei et al., 2020; Bodnar et al., 2022; Barbero et al., 2022). This covers citation networks (Sen et al., 2008; Namata et al., 2012) (Cora, Citeseer, Pubmed), webpages (WebKB) (Cornell, Texas, Wisconsin), actor co-occurrence networks (Tang et al., 2009) (Film), and Wikipedia networks (Rozemberczki et al., 2021) (Chameleon, Squirrel). Detailed description for each dataset can be found in Appendix C. This set of datasets contains graphs with varying sizes, densities, and homophily levels. A higher homophily level indicates that connected nodes are more likely to have similar features.

We use the 10 fixed splits in Pei et al. (2020) with 48%/32%/20% of nodes per class for training, validation, and testing, respectively. We follow the same hyperparameter settings in Bodnar et al. (2022) for each dataset. For our model-specific parameters, we use the 8 smallest eigenvectors as PEs, and a dimension $d = 3$ for the stalks in the sheaf. We use GCN as the base model, due to its well-known inability to deal with heterophilic graphs. We compare GCN with no PEs (**No PE**), with eigenvectors of the graph Laplacian as PEs (**GraphLap**), with eigenvectors of the precomputed sheaf Laplacian (**ConnLap**) and of the learnt sheaf Laplacian (**SheafLap**) as PEs. The reported results are the mean and standard deviation of accuracy (%) across the 10 folds.

	Texas	**Wisconsin**	**Film**	**Squirrel**	**Chameleon**	**Cornell**	**Citeseer**	**Pubmed**	**Cora**
Hom level	0.11	0.21	0.22	0.22	0.23	0.30	0.74	0.80	0.81
#Nodes	183	251	7,600	5,201	2,277	183	3,327	18,717	2,708
#Edges	295	466	26,752	198,493	31,421	280	4,676	44,327	5,278
#Classes	5	5	5	5	5	5	7	3	6
No PE	57.30±5.51	49.80±6.80	**25.20±0.69**	46.62±3.62	63.97±3.10	45.95±6.84	72.34±1.41	**86.43±0.35**	84.71±1.23
GraphLap	58.22±7.03	**55.49±12.46**	25.13±0.99	47.56±3.03	64.28±3.00	**51.35±7.15**	73.83±2.07	86.43±0.36	85.05±1.47
ConnLap	**58.38±7.76**	**57.65±6.63**	**26.53±0.86**	**47.92±3.53**	**65.57±2.52**	**52.97±7.37**	**73.88±1.84**	**86.49±0.42**	**85.13±1.34**
SheafLap	**61.08±6.19**	54.51±7.22	23.80±1.10	**51.11±2.95**	**65.2±3.10**	48.38±5.05	**74.35±1.64**	85.84±0.65	**85.88±1.26**

Table 1: Mean±std accuracy for node-level tasks with increasing homophily levels across 10 folds using different types of PEs. The best and the second best results are highlighted in **red** and **blue**, respectively.

Sheaf-based PEs outperform GraphLap Table 1 demonstrates that incorporating PEs, including GraphLap, ConnLap, and SheafLap, consistently improves performance over the baseline GCN (No PE) across all three datasets. While prior research (Dwivedi et al., 2020, 2021) primarily focused on graph-level tasks, we extend this evaluation to node-

level tasks. Notably, our sheaf-based PEs (ConnLap and SheafLap) outperform the graph Laplacian-based PE (GraphLap) in most scenarios. ConnLap excels across all tasks, and SheafLap obtains significant gains in most cases. This improvement aligns with our initial motivation, suggesting that the sheaf Laplacian can generate more expressive embeddings by incorporating node data, unlike the graph Laplacian, which relies solely on adjacency information. We provide additional ablation studies on the effect of normalisation, number of eigenvectors, qualitative visualisation, and complexity analysis in Appendix A.

Precomputed ConnLap vs. learnt SheafLap In Table 1, ConnLap consistently exhibits greater stability across datasets compared to the learnt sheaf Laplacian (SheafLap). This stability can be attributed to several factors. Firstly, the presence of a high feature dimension in these datasets provides rich information within the node features, which may support the underlying manifold assumption necessary for constructing ConnLap. Secondly, precomputed ConnLap avoids the numerical issues that can arise during backpropagation, a problem occasionally encountered by the learnt SheafLap. This contributes significantly to the improved stability of ConnLap. Lastly, it is worth noting that the datasets used in these experiments are relatively small. In such cases, the amount of available data may not be sufficient to effectively train a learnt sheaf.

4.2. Graph-level tasks

Graph-level tasks require mapping graphs to proper graph embeddings, which should be different for nonisomorphic graphs and identical for isomorphic ones. We use two sets of real-world molecular graphs, ZINC (Irwin et al., 2012) and OGBG-MOLTOX21 (Hu et al., 2020), for graph-level predictions. For the ZINC dataset, we use the predefined 10K/1K/1K splits for training, validation, and testing, respectively. For the OGBG-MOLTOX21 dataset, we adopt the default splits by OGB (Hu et al., 2020) based on scaffolding splitting (Wu et al., 2017). The hyperparameter setup follows Dwivedi et al. (2021), where GatedGCN, PNA, and SAN are used as base models. We select the 8 smallest eigenvectors as PEs, and a dimension $d = 3$ for the stalks in the sheaf. The reported results are the mean and standard deviation of metrics across 4 runs using different random seeds.

GatedGCN	**ZINC** TestMAE (↓)	**ZINC+LSPE** TestMAE (↓)	**MOLTOX21** TestAUC (↑)
No PE	0.251±0.009	N.A.	77.2±0.6
GraphLap	**0.202±0.006**	0.196±0.008	77.4±0.7
ConnLap	0.249±0.005	**0.193±0.014**	**77.9±0.2**

Table 2: Mean±std MAE (↓) for ZINC and mean±std AUC (↑) for MOLTOX21 across 4 random seeds using different types of PEs. The best result is highlighted in **red**.

	GatedGCN	**PNA**	**SAN**
	TestAUC (↑)		
No PE	77.2±0.6	**75.5±0.8**	74.4±0.7
GraphLap	77.4±0.7	75.2±1.3	73.6±0.3
ConnLap	**77.9±0.2**	75.3±0.4	**74.5±0.4**

Table 3: Mean±std AUC (↑) for MOLTOX21 across 4 random seeds using different base models. The best result is highlighted in **red**.

Precomputed ConnLap Due to the complexity of "unbatching" graphs during training, we focus on ConnLap for graph-level tasks in this study. We leave the efficient treatment of multi-graphs in sheaf learning for future work. Table 2 consistently shows that not using

positional encodings (No PE) yields the worst performance in both datasets, highlighting the effectiveness of Laplacian-based PEs. In MOLTOX21, ConnLap outperforms GraphLap (77.9 vs. 77.4), indicating that the sheaf Laplacian, considering both node data and graph structure, can provide more informative PEs at a global level. We also qualitatively show ConnLap captures the complex relationships between node data by visualising the constructed PEs for two random graphs from ZINC in Appendix A.4. We note that in the ZINC dataset, ConnLap only marginally improves (0.249 vs. 0.251) over No PE, while GraphLap excels (0.202). This discrepancy may stem from the sparse nature of node features in ZINC, where we use one-hot encodings to convert scalar values into feature vectors. The sparse information may not sufficiently satisfy the manifold assumption, unlike the richer 9-dimensional node features in MOLTOX21.

Learnable PEs To mitigate the problem, we allow the precomputed ConnLap to evolve during training using LSPE as explained in Section 3.1. The same LSPE mechanism is applied on GraphLap to ensure a fair comparison. In Table 2, under "ZINC+LSPE," GraphLap shows a slight improvement (0.202 to 0.196), while ConnLap benefits significantly from learning (0.249 to 0.193). This supports our hypothesis that the sheaf structure provides a solid foundation for ConnLap's evolution, and as node representations improve across layers, the connection Laplacian can better capture global positions from enhanced node features.

Different architectures PEs are also essential for enhancing Graph Transformers with graph structure input. Table 3 compares GraphLap and ConnLap across two GNN models (GatedGCN and PNA) and one Graph Transformer model (SAN), in line with previous research on graph Laplacian PEs (Dwivedi et al., 2021; Lim et al., 2022). ConnLap outperforms GraphLap and No PE with GatedGCN (77.9 vs. 77.4 and 77.2, respectively). However, for both PEs (75.2 and 75.3), they don't match the vanilla PNA architecture, indicating that Laplacian-based PEs may not be ideal for PNA's unique aggregator combination approach. We show in Appendix A.3 that PEs that capture the relative relationships between two nodes, such as random walks (Dwivedi et al., 2021), are better suited for PNA. Moving to SAN, a Graph Transformer, we see GraphLap underperforming (73.6) compared to No PE (74.4), while ConnLap offers a slight improvement (74.5) and higher stability (0.4). Overall, ConnLap consistently outperforms GraphLap in terms of accuracy and stability.

5. Conclusion

We introduce novel positional encodings (PEs) using sheaves, enhancing GNNs and Graph Transformers with both structural information and semantic knowledge from node data. Our experiments show that sheaf-based PEs outperform graph Laplacian-based methods, particularly in node differentiation. We explore precomputed and learnt sheaf PEs, striking a balance between computational complexity and expressive power. This work underscores the utility of sheaf theory in GNNs, emphasizing the value of geometric and topological concepts in advancing geometric deep learning.

References

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications, 2020. URL https://arxiv.org/abs/2006.05205.

Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković, and Pietro Liò. Sheaf neural networks with connection laplacians, 2022.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003. doi: 10.1162/089976603321780317.

Cristian Bodnar, Francesco Di Giovanni, Benjamin Paul Chamberlain, Pietro Liò, and Michael M. Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns, 2022. URL https://arxiv.org/abs/2202.04579.

J.-Y. Cai, M. Furer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. In *30th Annual Symposium on Foundations of Computer Science*, pages 612–617, 1989. doi: 10.1109/SFCS.1989.63543.

Justin Curry. Sheaves, cosheaves and applications, 2014.

B. L. Douglas. The weisfeiler-lehman method and graph isomorphism testing, 2011.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *CoRR*, abs/2012.09699, 2020. URL https://arxiv.org/abs/2012.09699.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks, 2020. URL https://arxiv.org/abs/2003.00982.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. 2021. doi: 10.48550/ARXIV.2110.07875. URL https://arxiv.org/abs/2110.07875.

Sergei Evdokimov and Ilia Ponomarenko. Isomorphism of coloured graphs with slowly increasing multiplicity of jordan blocks. *Combinatorica*, 19:321–333, 03 1999. doi: 10.1007/s004930050059.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation, 2019.

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

Jakob Hansen and Thomas Gebhart. Sheaf neural networks, 2020. URL https://arxiv.org/abs/2012.06333.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22118–22133. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/fb60d411a5c5b72b2e7d3527cfc84fd0-Paper.pdf.

John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. Zinc: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012. doi: 10.1021/ci3001277. URL https://doi.org/10.1021/ci3001277. PMID: 22587354.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention, 2021.

Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning, 2020. URL https://arxiv.org/abs/2009.00142.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 3538–3545. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16098.

Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning, 2022. URL https://arxiv.org/abs/2202.13013.

Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers, 2021.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks, 2018. URL https://arxiv.org/abs/1810.02244.

Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers, 2023.

Galileo Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven active surveying for collective classification. 2012.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer, 2023.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding, 2021.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/635440afdfc39fe37995fed127d7df4f-Paper.pdf.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008. doi: 10.1609/aimag.v29i3.2157. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2157.

Amit Singer and Hautieng Wu. Vector diffusion maps and the connection laplacian, 2011.

Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 807–816, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584959. doi: 10.1145/1557019.1557108. URL https://doi.org/10.1145/1557019.1557108.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Petar Veličković. Message passing all the way up. In *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*, 2022. URL https://openreview.net/forum?id=Bc8GiEZkTe5.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

WebKB. Cmu world wide knowledge base (web-¿kb) project. http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-11/www/wwkb/.

Zhenqin Wu, Bharath Ramsundar, Evan Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh Pappu, Karl Leswing, and Vijay Pande. Moleculenet: A benchmark for molecular machine learning. *Chemical Science*, 9, 03 2017. doi: 10.1039/C7SC02664A.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2018. URL https://arxiv.org/abs/1810.00826.

Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, Yan Song, Zhiguo Wang, and Dong Yu. Cross-lingual knowledge graph alignment via graph matching neural network. In *Proceedings of*

the 57th Annual Meeting of the Association for Computational Linguistics, pages 3156–3161, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1304. URL https://aclanthology.org/P19-1304.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation?, 2021.

Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks, 2019. URL https://arxiv.org/abs/1906.04817.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets, 2018.

Appendix A. Ablation Studies

A.1. Number of eigenvectors

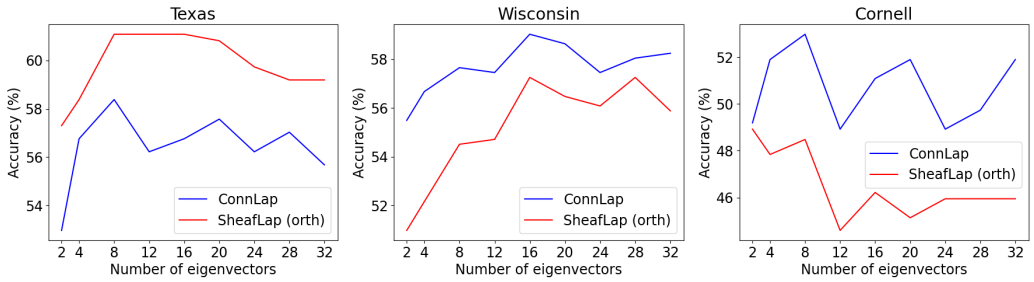

Figure 3: Accuracy (%) against number of eigenvectors used as PEs for ConnLap and SheafLap with an orthogonal matrix.

As shown in Table 3, using more eigenvectors improves the performance until a turning point is reached, beyond which the accuracy deteriorates. While more eigenvectors may exacerbate the sign ambiguity problem in eigendecompsition (Lim et al., 2022), an adequate number of eigenvectors is also needed to obtain sufficient positional information. We also note that this phenomenon is not observed in the Cornell dataset, suggesting that Laplacian-based PEs may not be well suited for this graph. This may explain the poor performance of SheafLap in Table 1.

A.2. Effect of normalisation

We compare the normalised and unnormalised sheaf Laplacian for constructing PEs using the learnt method (SheafLap). In Table 4, we observe that, except for one case, the unnormalised sheaf Laplacian proves to be a better candidate. The normalisation Laplacian matrix can reduce the number of sign possibilities (Dwivedi et al., 2020; Lim et al., 2022). Our results suggest that, in the case of the sheaf Laplacian, the benefit of normalisation to alleviate the sign ambiguity problem may not outweigh the expressive power of the unnormalised Laplacian in effectively differentiating nodes.

SheafLap	Texas		Wisconsin		Cornell	
	norm	unnorm	norm	unnorm	norm	unnorm
Diagonal	60.54±6.53	**61.35±6.63**	54.14±7.62	**54.90±9.80**	**44.86±4.55**	44.32±7.17
Orthogonal	59.19±6.89	**61.08±6.19**	54.49±7.44	**54.51±7.22**	46.49±5.90	**48.38±5.05**
General	59.02±7.34	**60.81±7.07**	55.89±6.81	**57.65±5.24**	45.26±4.31	**45.68±6.56**

Table 4: Mean±std accuracy (%) for node-level tasks comparing learnt sheaf PEs with normalised and unnormalised sheaf Laplacian. The winning case in each setup is highlighted in **bold**.

A.3. Comparison with relative PEs

	GatedGCN	PNA	SAN
No PE	77.2±0.6	75.5±0.8	74.4±0.7
RWPE	77.5±0.3	**76.1±0.7**	74.4±0.8
ConnLap	**77.9±0.2**	75.3±0.4	**74.5±0.4**

Table 5: Mean±std AUC for MOLTOX21 dataset across 4 random seeds comparing ConnLap with a relative positional encoding based on random walks (RWPE). The best result is highlighted in **red**.

In Table 5, we compare ConnLap, being a global PE, with a relative PE based on random walks (RWPE) (Dwivedi et al., 2021). We can see ConnLap maintains the best performance with GatedGCN (77.9 vs 77.5) and SAN (74.5 vs 74.4), except in the case of PNA (75.3 vs 76.1), in comparison with RWPE. We discussed previously that both global PEs using the Laplacian matrix tend to perform poorly with PNA. Here, we notice RWPE, as a relative PE, is more appropriate for this architecture. This supports our claim that PNA may not benefit from global structural knowledge, but relative distance information can facilitate its multiple aggregator scheme. Finally, we note that sheaf-based PEs can be transformed into relative PEs by taking the gradients of eigenvectors. We believe this leaves ample future avenues to explore.

A.4. Qualitative Analysis

We visualise the PEs generated from GraphLap and ConnLap by creating two synthetic graphs representing the chemical compounds decalin and bicyclopentyl, a well-known pair of nonisomorphic graphs that are hard to distinguish by GNNs (Sato et al., 2019). Node features are populated with alternating vectors of 0s and 1s to be heterophilic. PEs from GraphLap and ConnLap are visualised in Figure 4, where similar colours indicate similar PEs. We observe that GraphLap tends to generate a smoothly transiting colour scheme among neighbouring nodes, while ConnLap can generate dissimilar PEs when node features differ. For example, in decalin, GraphLap assigns either warm (red) or cold (blue) colours in each ring, while ConnLap assigns mixed colours. We show that ConnLap can effectively account for both structural and semantic information from the graph, generating a more expressive positional embedding space to facilitate node differentiation. Similar observation

is found in Figure 5, where we take two real-world graphs randomly from the OGBG-MOLTOX21 dataset (index 295 and 399).

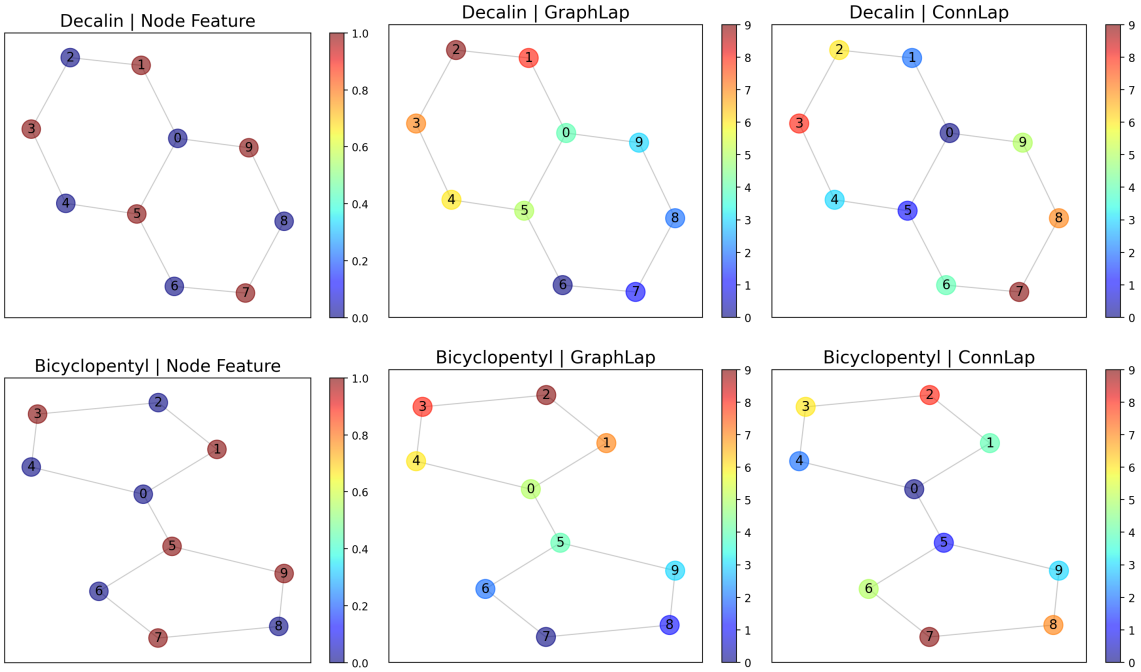

Figure 4: Visualisation of PEs created from GraphLap and ConnLap on two 10-node graphs with decalin and bicyclopentyl structures (Sato et al., 2019). These graphs are heterophilic, where nodes connected may have different node features.

A.5. Complexity Analysis

	Texas		**Citeseer**		**Squirrel**	
	precompute	per epoch	precompute	per epoch	precompute	per epoch
GraphLap	49.67	6.71	1886.58	4.47	9654.07	41.87
ConnLap	608.32	7.79	8562.46	3.48	72148.34	33.98
SheafLap	0	19.03	0	3139.45	0	31503.36

Table 6: Mean execution time in seconds (s) with different types of PEs on node-level tasks, sorted in increasing size of graph. In each dataset, "precompute" displays the total time cost to construct GraphLap/ConnLap during preprocessing time, and "per epoch" is the averaged time cost per epoch.

We compare the mean execution time for each type of PE on different datasets, trained on NVIDIA Tesla T4 GPU. In Table 6 and Table 7, we observe that the pre-computation time costs for ConnLap exceed those of GraphLap (around $\times 10$), due to the higher complexity for constructing the sheaf Laplacian than the graph Laplacian. However, this is a

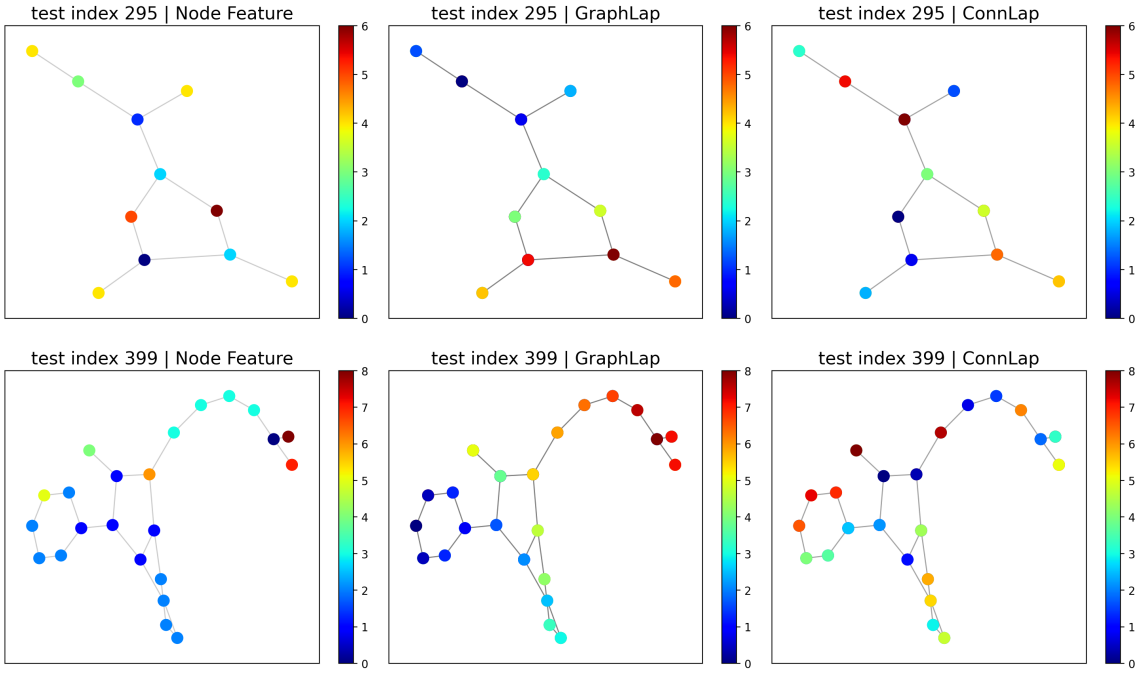

Figure 5: Visualisation of PEs created from GraphLap and ConnLap on 2 random graphs from OGBG-MOLTOX21 dataset.

	ZINC		**ZINC+LSPE**	**MOLTOX21**	
	precompute	per epoch	per epoch	precompute	per epoch
GraphLap	30.63	15.53	22.81	23.45	4.98
ConnLap	372.72	15.47	23.24	196.52	5.01

Table 7: Execution time in seconds (s) with different types of PEs on graph-level tasks, where ZINC+LSPE indicates allowing PEs to evolve during training.

one-off cost per dataset, and the time per epoch remains similar for both PEs. In Table 6, SheafLap induces a much higher cost per epoch (around $\times 1000$), which is significantly costlier considering that each training procedure typically requires hundreds of epochs. Although SheafLap does not require pre-computation, its high computational cost per epoch makes it less scalable than GraphLap and ConnLap.

Appendix B. Precomputed connection Laplacian

We provide the details to compute the parallel transport from $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ to $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$ on the manifold $\mathcal{M}$ in the context of graph data as proposed in (Barbero et al., 2022).

The two-step procedure is summarised in Figure 6. Here, we elaborate the steps in more details. First, we construct the orthonormal bases of the tangent spaces $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ for each data point $\mathbf{x}_v$ via local Principle Component Analysis (PCA). The local neighbourhood is defined by the 1-hop neighbourhood $\mathcal{N}(v)$ of the node v, giving us a $p \times |\mathcal{N}(v)|$ matrix

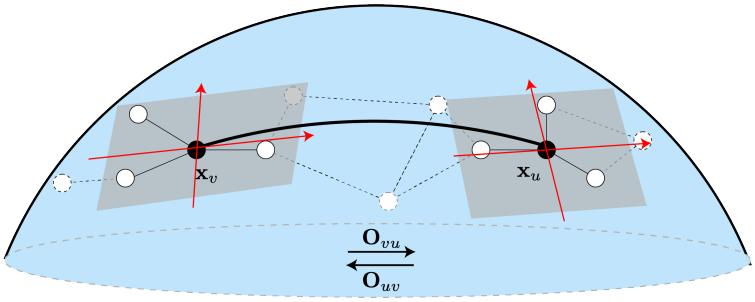

Figure 6: Step 1: We construct the orthonormal bases of $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ and $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$ using local PCA, defined by the 1-hop neighbourhood of the nodes. Step 2: We compute the orthogonal mapping $\mathbf{O}_{vu}$ by optimal aligning the two bases, which approximates the parallel transport operator (Singer and Wu, 2011).

$\hat{\mathbf{X}}_v = [\mathbf{x}_v - \mathbf{x}_v, ..., \mathbf{x}_{v_{|\mathcal{N}(v)|}} - \mathbf{x}_v]$. We perform PCA via Singular Value Decomposition (SVD) on the matrix $\hat{\mathbf{X}}_v = \mathbf{U}_v \sum_i \mathbf{V}_v^\top$. The stalk dimension d is set as a hyperparameter, which is also the dimension of the orthonormal bases. Therefore, we take the first d left singular vectors of $\mathbf{U}_v$. This forms a d-dimensional subspace of $\mathbb{R}^p$, calling it $\mathbf{O}_v$. We take $\mathbf{O}_v$ as an approximation for the basis of the tangent space $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$.

Next, we compute the parallel transport via optimal alignment between the tangent spaces $\mathcal{T}_{\mathbf{x}_v}\mathcal{M}$ and $\mathcal{T}_{\mathbf{x}_u}\mathcal{M}$. Intuitively, an optimal alignment is a mapping from one tangent space to another. Formally, we compute $\mathbf{O}_{vu} = \mathbf{U}\mathbf{V}^\top$, where $\mathbf{U}$ and $\mathbf{V}$ comes from the SVD of $\mathbf{O}_v^\top \mathbf{O}_u = \mathbf{U}\sum\mathbf{V}^\top$. If $\mathbf{x}_v$ and $\mathbf{x}_u$ are close enough, then $\mathbf{O}_{ij}$ approximates the parallel transport between their tangent spaces as proved in (Singer and Wu, 2011). This condition is ensured by taking the 1-hop neighbourhood as explained in the last paragraph. Note when there are fewer than d neighbours for node v, we take the nearest nodes according to the Euclidean distances of their node data.

Appendix C. Description of datasets

C.1. Node-level tasks

Citation networks (Cora, Citeseer, Pubmed) In these citation networks (Sen et al., 2008; Namata et al., 2012), nodes are academic papers with edges showing their citation relationships. The node features are the bag-of-words representation of the papers. The goal is to predict the academic topic of each paper.

WebKB (Cornell, Texas, Wisconsin) These are webpage datasets (WebKB) collected from computer science departments in various universities. The nodes are webpages, while the edges are hyperlinks between them. Similarly, the node features are the bag-of-words representation, with the goal to classify the category of the webpage.

Actor co-occurrence network (Film) This dataset (Tang et al., 2009) is a subgraph of the film-director-actor-writer network, with nodes representing actors and edges denoting co-occurrence on the same Wikipedia page. The node features consist of keywords in the Wikipedia page, while the node label corresponds to the actor's Wikipedia category.

Wikipedia network (Chameleon, Squirrel) The two Wikipedia networks (Rozemberczki et al., 2021) consist of pages related with chameleon or squirrel. The nodes are Wikipedia pages, and the edges are mutual inks between them. Node features contain information nouns from the page. The task requires to classify the pages into five categories based on their average monthly traffic.

C.2. Graph-level tasks

ZINC The ZINC dataset contains 12K molecular graphs representing commercially available chemical compounds. The molecular graphs vary in size, from 9 to 37 nodes. Each node or edge is given a number as the feature, representing its atom or bond type. In total, there are 28 atom types and 3 bond types. This graph regression task requires the model to predict a constrained solubility (logP) value of the compound.

OGBG-MOLTOX21 This dataset is a smaller dataset taken from MoleculeNet (Wu et al., 2017). The nodes are atoms with 9-dimensional vector features, encoding information such as atomic number, chirality, and formal charge. The edges represent chemical bonds. It is a graph-level classification task with binary labels.