HIERARCHICAL POOLING FOR SHEAF NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sheaf Neural Networks (SNNs) extend Graph Neural Networks (GNNs) by operating on cellular sheaves, which assign to each node and edge its own vector space, along with linear restriction maps from node spaces to the spaces associated with their incident edges. These maps determine the sheaf Laplacian and can be learned at each layer, enabling layer-specific diffusion dynamics. Existing SNN architectures, however, can only operate at a fixed graph resolution. We introduce Hierarchical Sheaf Pooling (HiSP), a pooling layer that enables hierarchical representation learning in SNNs. Following the Select–Reduce–Connect (SRC) framework, HiSP groups fine-scale nodes, reduces their features using the fine-scale sheaf structure, and constructs the corresponding coarse graph. A new sheaf is then learned at the coarse resolution. By interleaving sheaf diffusion and pooling, HiSP enables hierarchical SNNs to learn distinct diffusion dynamics across layers and graph resolutions. This construction supports multiscale problems in which aggregation changes both the interacting entities and their relations.

1 INTRODUCTION

GNNs update node representations by repeatedly aggregating information from neighboring nodes (Gori et al., 2005; Scarselli et al., 2008). Many widely used GNN layers can be interpreted as diffusion or filtering operators derived from the graph Laplacian, which encodes graph connectivity and penalizes differences between adjacent node representations (Defferrard et al., 2016; Kipf & Welling, 2017). SNNs generalize this mechanism using cellular sheaves, which assign a vector space, called a stalk, to each vertex and edge, together with restriction maps that send vertex data to the stalks of its incident edges (Hansen & Gebhart, 2020; Bodnar et al., 2022; Barbero et al., 2022b;a; Zaghen et al., 2024; Braithwaite et al., 2024; Purificato et al., 2025; Borgi et al., 2025). These maps determine the sheaf Laplacian, whose diffusion promotes agreement after neighboring representations have been mapped to a common edge stalk. When the restriction maps are inferred from hidden representations, different layers can learn different notions of compatibility and, consequently, different diffusion dynamics (Bodnar et al., 2022).

Existing SNN architectures operate at a fixed graph resolution: restriction maps may change across layers, but they continue to describe interactions between nodes of the original graph (Bodnar et al., 2022). Pooling can create a hierarchy of entities, allowing subsequent layers to model interactions specific to each resolution. This is essential when scale changes both the interacting entities and their relations. In opinion dynamics, for example, fine-scale restriction maps describe how individuals express private opinions in interpersonal discourse spaces (Hansen & Ghrist, 2021). A pooled node may represent the aggregate position of a social group or a political party. Relations between these coarse entities may encode institutional mechanisms such as coalition, negotiation, or opposition, which differ from interpersonal communication.

Hierarchical GNNs address multiscale structure by repeatedly dropping nodes or grouping them into supernodes, thereby constructing local summaries at progressively lower spatial resolutions (Grattarola et al., 2022). This process expands the receptive field of a GNN, reduces the computational graph, and progressively distills local information into global representations. Several pooling operators assign nodes to a cluster and aggregate their representations in a common feature space (Ying et al., 2018; Bianchi et al., 2019; Grattarola et al., 2022; Abate & Bianchi, 2025). In an SNN, however, each node representation belongs to a distinct vertex stalk (Hansen & Ghrist, 2019). Even when

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

all stalks have the same dimension, their coordinates might have different semantics, so aggregating them as if they lived in a shared Euclidean feature space is not well-defined. A sheaf-aware pooling layer must additionally define how representations from different stalks are related, aggregated, and represented on the coarse graph.

We introduce *Hierarchical Sheaf Pooling (HiSP)*, the first sheaf-aware hierarchical pooling operator that progressively coarsens a sheaf by clustering nodes defined in different vertex stalks. Within each cluster, HiSP uses the learned restriction maps to construct *transport maps* that project node representations into a common reference stalk. The aligned representations are then aggregated into a single coarse-node representation. The resulting coarse topology is passed to a subsequent sheaf layer, which learns new restriction maps between coarse nodes. The sheaf structure and its associated diffusion dynamics can thus adapt to the resolution produced by pooling.

We further provide the components required for end-to-end graph-level learning with SNNs in Pytorch Geometric (PyG). Specifically, we implement neural sheaf diffusion (Bodnar et al., 2022; Fey & Lenssen, 2019), integrate HiSP into the modular Select–Reduce–Connect (SRC) framework and the Torch Geometric Pool (`tgp`) workflow (Grattarola et al., 2022; Abate et al., 2026). We also introduce a sheaf-aware graph readout. Together with HiSP, these components provide a complete PyG workflow for hierarchical SNNs on graph-level tasks. The code is available at `https://anonymous.4open.science/r/hisp/`.

2 Background and Related Work

2.1 Cellular sheaves and sheaf Laplacians

Let $G = (V, E)$ be a graph with $N = |V|$ vertices. A *cellular sheaf* $\mathcal{F}$ on G assigns a finite-dimensional inner-product space $\mathcal{F}(v)$ to each vertex v, and $\mathcal{F}(e)$ to each edge e, and a linear map $\rho_{e,v} \coloneqq \mathcal{F}_{v \trianglelefteq e} : \mathcal{F}(v) \longrightarrow \mathcal{F}(e)$ for every incident pair $v \trianglelefteq e$ (Hansen & Ghrist, 2019; 2021). The vector spaces $\mathcal{F}(v)$ and $\mathcal{F}(e)$ are called *stalks*, while the maps $\mathcal{F}_{v \trianglelefteq e}$ are referred to as *restriction maps*. A 0-*cochain* is a choice of a vector $x_v \in \mathcal{F}(v)$ for each $v \in V$; a 1-*cochain* is a choice of $x_e \in \mathcal{F}(e)$. The spaces of 0- and 1-cochains are the direct sum of the stalks over vertices and edges, respectively, i.e., $C^0(G;\mathcal{F}) = \bigoplus_{v \in V} \mathcal{F}(v)$ and $C^1(G;\mathcal{F}) = \bigoplus_{e \in E} \mathcal{F}(e)$, and are equipped with the corresponding direct-sum inner products. After fixing an arbitrary edge orientation, the coboundary operator $\delta_{\mathcal{F}} : C^0(G;\mathcal{F}) \to C^1(G;\mathcal{F})$ measures the disagreement between the endpoint restrictions and is defined by $(\delta_{\mathcal{F}} x)_e = \rho_{e,u} x_u - \rho_{e,v} x_v$ for every oriented edge $e = (u, v)$. The associated *combinatorial sheaf Laplacian* is

$$L_{\mathcal{F}} \coloneqq \delta_{\mathcal{F}}^* \delta_{\mathcal{F}} : C^0(G;\mathcal{F}) \longrightarrow C^0(G;\mathcal{F}), \tag{1}$$

where $\delta_{\mathcal{F}}^*$ denotes the adjoint with respect to the direct-sum inner products. $L_{\mathcal{F}}$ is self-adjoint, positive semidefinite, and independent of the chosen edge orientation. Its quadratic form

$$\langle x, L_{\mathcal{F}} x \rangle = \|\delta_{\mathcal{F}} x\|^2 = \sum_{e=(u,v) \in E} \|\mathcal{F}_{u \trianglelefteq e} x_u - \mathcal{F}_{v \trianglelefteq e} x_v\|^2, \tag{2}$$

measures the total disagreement between adjacent node representations in their common edge stalks.

2.2 Sheaf Neural Networks

Let $D_0 = \dim C^0(G;\mathcal{F})$. Here, we assume that all vertex and edge stalks have the same dimension $\mathbb{R}^d$. Therefore, $D_0 = Nd$. An h-channel sheaf signal is represented by a matrix $\mathbf{X} \in \mathbb{R}^{Nd \times h}$, whose columns are 0-cochains. SNNs propagate these signals using a diffusion operator induced by the sheaf Laplacian. The underlying diffusion reduces disagreement between neighboring representations after they have been mapped to their shared edge stalks, rather than directly comparing their vertex coordinates. A sheaf convolution augments this process with learnable transformations of the stalk coordinates and feature channels, followed by a nonlinear activation (Hansen & Gebhart, 2020). Neural Sheaf Diffusion (NSD) extends this construction by learning a layer-dependent sheaf from the current hidden representations (Bodnar et al., 2022). At layer ℓ, let $\mathbf{X}_u^{(\ell)} \in \mathbb{R}^{d \times h}$ denote the block of $\mathbf{X}^{(\ell)}$ associated with vertex u. For an edge $e = \{u, v\}$, the endpoint restriction maps are parameterized as

$$\mathcal{F}_{u \trianglelefteq e}^{(\ell)} = \Phi_{\theta_\ell}\left(\mathbf{X}_u^{(\ell)}, \mathbf{X}_v^{(\ell)}\right), \qquad \mathcal{F}_{v \trianglelefteq e}^{(\ell)} = \Phi_{\theta_\ell}\left(\mathbf{X}_v^{(\ell)}, \mathbf{X}_u^{(\ell)}\right), \tag{3}$$

where Φ_{θ_ℓ} is a matrix-valued function implemented by an MLP with trainable parameters θ_ℓ followed by reshaping. Its outputs may be parameterized as general, diagonal, or orthogonal matrices. Thus, the restriction maps, and consequently the diffusion operator, may change across layers. Let $\mathbf{D}^{(\ell)}$ denote the block-diagonal part of $\mathbf{L}_{\mathcal{F}^{(\ell)}}$. The normalized sheaf Laplacian is $\Delta_{\mathcal{F}^{(\ell)}} := \left(\mathbf{D}^{(\ell)}\right)^{-1/2} \mathbf{L}_{\mathcal{F}^{(\ell)}} \left(\mathbf{D}^{(\ell)}\right)^{-1/2}$. The corresponding residual update is

$$\mathbf{X}^{(\ell+1)} = \mathbf{X}^{(\ell)} - \sigma\left[\Delta_{\mathcal{F}^{(\ell)}} \left(\mathbf{I}_N \otimes \mathbf{W}_1^{(\ell)}\right) \mathbf{X}^{(\ell)} \mathbf{W}_2^{(\ell)}\right]. \tag{4}$$

Here, σ denotes a nonlinear activation, $\mathbf{W}_1^{(\ell)} \in \mathbb{R}^{d\times d}$ mixes coordinates within each vertex stalk, while $\mathbf{W}_2^{(\ell)} \in \mathbb{R}^{h\times h}$ mixes feature channels.

2.3 Hierarchical Graph Pooling

Let $\mathbf{A} \in \mathbb{R}_+^{N\times N}$ be the adjacency matrix of G, and let $\mathbf{H} \in \mathbb{R}^{N\times h}$ be its node-feature matrix, where $\mathbf{H}_v \in \mathbb{R}^h$ denotes the representation of vertex v. A pooling layer maps the graph $(\mathbf{A}, \mathbf{H})$ to a coarser representation with $K \leq N$ nodes: $\texttt{POOL} : (\mathbf{A}, \mathbf{H}) \longmapsto (\mathbf{A}', \mathbf{H}')$, where $\mathbf{A}' \in \mathbb{R}^{K\times K}$ and $\mathbf{H}' \in \mathbb{R}^{K\times h}$. We describe graph pooling through the *SRC* framework (Grattarola et al., 2022), which decomposes the pooling operation into three functions. First, the selection function SEL assigns the N input nodes to K *supernodes* and returns an assignment matrix according to $\texttt{SEL} : (\mathbf{A}, \mathbf{H}) \mapsto \mathbf{S} \in \mathbb{R}^{N\times K}$, where S_{ia} denotes the membership of node i in supernode a. Membership can either be "hard" (each node is assigned to exactly one supernode) or "soft" (each node can belong to multiple supernodes with different membership scores, typically normalized to sum to one). Then, the reduction function RED aggregates the features of the nodes assigned to each supernode: $\texttt{RED} : (\mathbf{H}, \mathbf{S}) \mapsto \mathbf{H}'$. A standard sum-based reduction is $\mathbf{H}' = \mathbf{S}^\top \mathbf{H}$. Finally, the connection function CON constructs the connectivity of the pooled graph: $\texttt{CON} : (\mathbf{A}, \mathbf{S}) \mapsto \mathbf{A}'$. A common choice is $\mathbf{A}' = \mathbf{S}^\top \mathbf{A}\mathbf{S}$. Torch Geometric Pool (tgp) is a pooling library that implements this decomposition through a modular interface in which the SEL, RED, and CON components can be defined and combined independently.

3 Hierarchical Sheaf Pooling

3.1 Problem formulation and method overview

The input consists of a graph $G = (V, E)$, a learned sheaf $\mathcal{F}$, and node features $\mathbf{X}$. The pooling operator produces a coarser graph with $K < |V|$ nodes: $\texttt{POOL} : \left(G, \mathcal{F}, \mathbf{X}\right) \longmapsto \left(G', \mathbf{X}'\right)$. Unlike ordinary graph features, node features belong to distinct stalk spaces and therefore cannot be aggregated directly. HiSP partitions the nodes into internally connected clusters and equips each cluster C_a with a rooted spanning tree that connects each node to its representative r_a (the root). Using the restriction maps of $\mathcal{F}$, it constructs transport maps along the tree and aligns each node feature with the root stalk. The aligned features are then aggregated into a single coarse-node representation, while inter-cluster connectivity defines the coarse graph:

$$\left(G, \mathcal{F}, \mathbf{X}\right) \xrightarrow{\texttt{SEL}} \left\{r_a\right\}_{a=1}^{K} \xrightarrow{\texttt{RED}} \mathbf{X}' \xrightarrow{\texttt{CON}} G'.$$

The restriction maps on G' are then learned by the subsequent sheaf layer. An illustrative overview of the method is provided in Figure 1.

3.2 Select

The SELECT stage partitions the vertex set into K connected clusters, $V = C_1 \sqcup \cdots \sqcup C_K$. For each cluster C_a, it chooses a root $r_a \in C_a$, which represents the cluster as a *supernode* in the coarse graph. Its stalk $\mathcal{F}(r_a)$ serves as the reference stalk for aggregating the features in C_a. The coarse vertex set is therefore $V' = \{r_a\}_{a=1}^K$. The partition is encoded by the hard assignment matrix $\mathbf{S} \in \{0,1\}^{N\times K}$, with $S_{va} = \mathbf{1}[v \in C_a]$. HiSP is independent of the selection strategy; it only requires that a rooted spanning tree can be constructed within each cluster.

We consider two topology-based strategies, chosen because their assignments depend only on the fixed graph topology and can therefore be precomputed before training. k-Maximal Independent

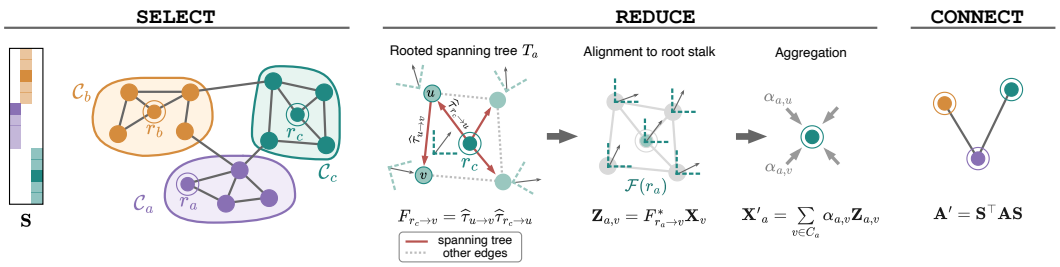


Figure 1: HiSP overview. SELECT partitions the graph into connected clusters and chooses a root for each cluster. REDUCE aligns node features in the corresponding root stalk through tree-based transport and aggregates them into a coarse-node representation. CONNECT constructs the coarse graph from the cluster assignments.

Sets pooling (k-MIS) selects well-separated representatives (Bacciu et al., 2023) and grows clusters around them using multi-source Breadth-First Search (BFS), repeatedly bisects connected subgraphs using their Fiedler vectors (Pothen et al., 1990) and selects the maximum-degree node within each final cluster as its representative. Both methods produce connected clusters for REDUCE; their construction and partition properties are detailed in Appendix A.

The SELECT stage returns the assignment matrix together with the clusters and their roots:

$$\mathrm{SEL}(G) = \left(\mathbf{S}, \{r_a\}_{a=1}^{K}\right). \tag{5}$$

3.3 REDUCE

3.3.1 TRANSPORT MAPS

Let $G[C_a]$ denote the subgraph of G induced by C_a. Because C_a is connected, $G[C_a]$ admits a spanning tree T_a, which we root at r_a. Each node $v \in C_a \setminus \{r_a\}$ has a unique parent $p_a(v)$ and a unique path from r_a to v. Consider a tree edge $e = \{u, v\}$, oriented from the parent $u = p_a(v)$ to the child v. The restriction maps $\rho_{e,u}$ and $\rho_{e,v}$ map both endpoint stalks to the common edge stalk. Transporting a feature from u to v therefore requires mapping $\rho_{e,u}\mathbf{x}_u$ back through $\rho_{e,v}$. However, $\rho_{e,v}$ may not admit an inverse. We instead seek a vector in $\mathcal{F}(v)$ whose restriction approximates $\rho_{e,u}\mathbf{x}_u$, using a Tikhonov-regularized inverse (Hansen, 1998) and define

$$\tau_{u\to v} = \left(\rho^*_{e,v}\rho_{e,v} + \varepsilon\mathbf{I}\right)^{-1}\rho^*_{e,v}\rho_{e,u} : \mathcal{F}(u) \longrightarrow \mathcal{F}(v), \qquad \varepsilon > 0. \tag{6}$$

This is the regularized least-squares transport that minimizes the disagreement between the parent and child representations in the common edge stalk. The regularization term ε makes the inverse well-defined and stabilizes it when $\rho_{e,v}$ is singular or ill-conditioned. Even with regularization, $\tau_{u\to v}$ may have operator norm greater than one. To keep feature norms from growing along tree paths, we normalize each transport as $\widehat{\tau}_{u\to v} = \tau_{u\to v}/\max\{1, \|\tau_{u\to v}\|_2\}$, so that $\|\widehat{\tau}_{u\to v}\|_2 \le 1$: feature norms cannot grow along a path, though they may shrink with repeated application.

To transport a feature from the root r_a to a node v, we apply the normalized maps in order along the unique tree path between them. The resulting map $F_{r_a\to v} : \mathcal{F}(r_a) \to \mathcal{F}(v)$ can be built one edge at a time:

$$F_{r_a\to r_a} = \mathbf{I}_d, \qquad F_{r_a\to v} = \widehat{\tau}_{p_a(v)\to v}F_{r_a\to p_a(v)}. \tag{7}$$

At the root, the feature is unchanged; to reach any other node v, we first transport it to its parent $p_a(v)$, then apply the map from the parent to v. Reduction uses the adjoint of the root-to-node transport. For $F_{r_a\to v} : \mathcal{F}(r_a) \to \mathcal{F}(v)$, $\langle F_{r_a\to v}g, \mathbf{x}_v\rangle_v = \langle g, F^*_{r_a\to v}\mathbf{x}_v\rangle_{r_a}$. Thus $F^*_{r_a\to v}\mathbf{x}_v$ is the contribution of the feature at v to root-stalk test features under the lift. It is the dual reduction operator: the same pairing that defines the lift, evaluated in the root stalk. We define its aligned contribution as

$$\mathbf{Z}_{a,v} = F^*_{r_a\to v}\mathbf{X}_v \in \mathcal{F}(r_a)^h \simeq \mathbb{R}^{d\times h}. \tag{8}$$

Therefore, all features $\{\mathbf{Z}_{a,v} : v \in C_a\}$ belong to the same stalk and can be aggregated. Further details on the construction of the transport maps and an analysis of pathwise amplification are provided in Appendix B.

3.3.2 TRANSPORT MEAN

Once the node features have been aligned in the root stalk, we compute for each node a scalar score $s_{a,v} = s_\theta(\mathrm{vec}(\mathbf{Z}_{a,v}))$, where s_θ is a learnable scoring function. The scores are normalized within each cluster $\alpha_{a,v} = \exp(s_{a,v}) / \sum_{u \in C_a} \exp(s_{a,u})$, so that $\sum_{v \in C_a} \alpha_{a,v} = 1$. The updated feature assigned to the supernode r_a is the weighted transport mean:

$$\mathbf{X}_{c,a} = \sum_{v \in C_a} \alpha_{a,v} \mathbf{Z}_{a,v} = \sum_{v \in C_a} \alpha_{a,v} F^*_{r_a \to v} \mathbf{X}_v \in \mathcal{F}(r_a)^h.$$

Thus, the cluster is represented by a convex combination of its node features after they have been mapped to the common root stalk. The collected representations of all clusters is $\mathbf{X}' = \{\mathbf{X}_{c,a}\}_{a=1}^K$ and the REDUCE output is

$$\mathtt{RED}_T\left(\mathcal{F}, \mathbf{X}, \{C_a, r_a\}_{a=1}^K\right) = \mathbf{X}'(T), \tag{9}$$

where $T = \{T_a\}_{a=1}^K$ denotes the set of rooted spanning trees used for transport.

3.3.3 PATH DEPENDENCE

Within a cluster, two different paths π_1 and π_2 may connect the root r_a to the same node v. Each path applies a different sequence of edge transport maps, and the learned restriction maps do not guarantee that these sequences give the same result. The resulting transport maps may therefore differ: $F^{(\pi_1)}_{r_a \to v} \neq F^{(\pi_2)}_{r_a \to v}$. Consequently, the aligned feature $F^{(\pi)*}_{r_a \to v} \mathbf{X}_v$, and hence the resulting coarse-node representation, may depend on the path selected within the cluster. The spanning tree chosen for each cluster determines these paths. We optionally resample BFS spanning trees during training, exposing the model to multiple transport paths rather than a single fixed realization. This does not remove path dependence, but trains the model over a distribution of valid trees. A detailed analysis and the corresponding stochastic objective are provided in Appendix B.

3.4 CONNECT

The CONNECT stage defines the edges of the coarse graph G' using the assignment matrix $\mathbf{S}$ from Equation (5):

$$\mathtt{CON}(\mathbf{A}, \mathbf{S}) = \mathbf{A}', \qquad \mathbf{A}' = \mathbf{S}^\top \mathbf{A} \mathbf{S}. \tag{10}$$

For two distinct supernodes, the edge weight is the sum of the original edge weights between their clusters. The restriction maps on G' are learned by the subsequent sheaf layer.

3.5 HISP READOUT

For graph-level tasks, alternating the sheaf-diffusion layers of Equation (4) with HiSP layers produces a sequence of progressively coarser graphs $G_0 \longrightarrow G_1 \longrightarrow \cdots \longrightarrow G_L$, where $G_\ell = (V_\ell, E_\ell)$. The complete graph- and node-level architectures are described in Appendix C. The final graph G_L may contain multiple supernodes, either because coarsening stops before reaching a single node or because the input graph is disconnected. Their features belong to distinct root stalks and therefore cannot be averaged directly. Instead, we extract descriptors from each supernode, map them to common Euclidean embedding spaces, and aggregate them there.

For each final supernode $r_a \in V_L$, representing a cluster $C_a \subseteq V_{L-1}$, we compute three summaries in the chosen stalk coordinates; the stalk-coordinate mean retains dependence on that coordinate choice. The channel Gram descriptor $\boldsymbol{\phi}_a^{(\mathrm{Gram})} = \mathbf{X}_{r_a}^{(L)\top} \mathbf{X}_{r_a}^{(L)}$ records inner products between feature channels. The stalk-coordinate mean $\boldsymbol{\phi}_a^{(\mathrm{mean})} = d^{-1} \mathbf{1}_d^\top \mathbf{X}_{r_a}^{(L)}$ averages each channel over the d stalk coordinates. The cluster-statistics descriptor collects the pooled magnitude, the sheaf Dirichlet energy inside the cluster, and the magnitude before pooling:

$$\boldsymbol{\phi}_a^{(\mathrm{stat})} = \left(\left\| \mathbf{X}_{r_a}^{(L)} \right\|_{\mathrm{F}}, \ \mathrm{tr}\left(\left(\widetilde{\mathbf{X}}_{C_a}^{(L-1)} \right)^\top \mathbf{L}_a^{(L-1)} \widetilde{\mathbf{X}}_{C_a}^{(L-1)} \right), \ \left\| \widetilde{\mathbf{X}}_{C_a}^{(L-1)} \right\|_{\mathrm{F}} \right).$$

Here, $\widetilde{\mathbf{X}}_{C_a}^{(L-1)}$ stacks the cluster's node features before the final pooling step, and $\mathbf{L}_a^{(L-1)}$ is the sheaf Laplacian restricted to edges within C_a. For each descriptor type $b \in \mathcal{B} = \{\mathrm{Gram}, \mathrm{mean}, \mathrm{stat}\}$, a

learnable MLP ψ_b, shared across all coarse nodes, maps the descriptor to a branch embedding; the Gram matrix is first flattened into a vector. These embeddings are averaged over the final coarse nodes and concatenated:

$$\mathbf{r}_G^{(b)} = \frac{1}{|V_L|} \sum_{a \in V_L} \psi_b\left(\boldsymbol{\phi}_a^{(b)}\right), \quad \mathbf{g}_G = \Big\Vert_{b \in \mathcal{B}} \mathbf{r}_G^{(b)}, \quad \widehat{\mathbf{y}}_G = \mathrm{MLP}_{\mathrm{out}}(\mathbf{g}_G). \tag{11}$$

4 Experiments

We evaluate whether a our proposed Hierarchical Sheaf Neural Network (HSNN) can: (i) learn independent sheaves across resolutions; (ii) transmit sheaf signals over long graph distances; (iii) filter within-cluster distractors; (iv) improve long-range node prediction; (v) transfer the hierarchy to real community graphs; and (vi) support sheaf U-Nets and graph classification on standard benchmarks. Finally, we ablate the `REDUCE` and `READOUT` components. Unless otherwise stated, results are reported as mean $\pm$ standard deviation over five random seeds.

Baselines. We compare against NSD (Bodnar et al., 2022) as a fixed-resolution sheaf baseline and GCN and GIN as standard message-passing baselines. To test whether a Euclidean hierarchy suffices, we equip GCN and GIN with EigenPooling (Ma et al., 2019), denoted simply as *pool* in the tables (e.g., GCN+pool). Dataset construction and search spaces are deferred to Appendix D.

4.1 Independent sheaves across resolutions

A HSNN should learn a fine sheaf that is locally consistent inside communities and a coarse sheaf that tests a different compatibility after pooling. The point of these tasks is not that message passing is weak in general, but that the *label lives at a coarser sheaf resolution* than the edges the model sees: without an explicit hierarchy, a single fine sheaf has no place to represent the coarse constraint. We probe this with the following graph-classification task.

Hierarchical BIpartite (HierBI). Each graph has four rooms connected by corridors. At the fine resolution, every room is a bipartite graph $K_{8,8}$. Its two partitions encode the same room-level content with opposite feature signs. Consequently, averaging the node features within a room cancels the room-level signal. Pooling each room into one supernode produces the coarse graph $K_{2,2}$. Every room is assigned a hidden sign $s_c \in \{+, -\}$, which determines its room-level content but is not provided to the model as an explicit feature. Writing the four rooms in a fixed order, the graph receives label A or B according to the corresponding sign pattern:

$$\text{A:} \quad (+,+,-,-), \qquad \text{B:} \quad (+,-,+,-).$$

The construction is applied independently to two feature channels, and their pair of patterns defines one of four balanced classes: AA, AB, BA, or BB. Chance accuracy is therefore 25%.

Solving HierBI involves two resolutions. Within each room, the oppositely signed features must first be aligned before aggregation. The resulting four room representations must then be compared on the coarse graph to identify their joint pattern. Full construction and training details are provided in Appendix D.2. As shown in Table 1, NSD+HiSP reaches 92.7% accuracy, whereas NSD and both Pool baselines remain below 50%. This gap supports the benefit of using the fine sheaf to construct room-level representations and learning a new sheaf to model their coarse relations. Section 4.7 further examine the contributions of the reduction and readout components.

Table 1: HierBI test accuracy (%, mean $\pm$ std; $\uparrow$).

Model	GCN+pool	GIN+pool	NSD	NSD+HiSP
Accuracy	48.4±4.3	42.8±2.5	47.8±3.0	**92.7±1.2**

4.2 Long-range signal transfer

RingTransfer (Bodnar et al., 2021) tests whether a model can transmit a class label across a chordless 24-cycle. The label is provided at a source node and must be predicted at the antipodal query

node, which is 12 edges away. The task has five classes, giving a chance accuracy of 20%. NSD can solve the task when its depth covers the distance between the source and query. However, NSD ($L \leq 6$) remains at chance, while 12-layer NSD reaches perfect accuracy only for some random seeds. HiSP reduces the source–query distance from 12 on the original cycle to 2 on the coarse graph. Consequently, NSD+HiSP solves the task consistently with only $L = 3$ layers. This experiment shows that hierarchical coarsening can reduce the diffusion depth required to transmit a sheaf signal over long graph distances. More details are provided in Appendix D.5.

Table 2: RingTransfer test accuracy (%, mean ± std; ↑). NSD depth ladder vs. one-stage HiSP (K=6). GCN/GIN+pool remain at chance under matched cards.

Model	GCN/GIN+pool	NSD (L=3)	NSD (L=6)	NSD (L=12)	NSD+HiSP (L=3)
Accuracy	20.0±0.0	20.0±0.0	20.0±0.0	72.0±34.9	**100.0±0.0**

4.3 Filtering capacity inside clusters

Marked PathCopy. We test whether HiSP can preserve a relevant signal while coarsening distractor-rich regions. Each graph contains a path whose endpoints are a source and a query node. The source stores one of five class labels, which the query must recover, while every second internal node carries an incorrect label. Dataset construction and training details are provided in Appendix D.4. The restriction maps are learned end-to-end through both sheaf diffusion and transport-based reduction. Thus, this experiment evaluates whether the joint hierarchical operator can learn to propagate the source signal while suppressing distractors. As shown in Table 3, NSD reaches 55.5%, while NSD+HiSP reaches 87.5% and outperforms both standard message-passing and Euclidean hierarchical baselines.

Table 3: Marked PathCopy test accuracy (%, mean ± std.; ↑).

Model	GCN	GIN	GCN+pool	GIN+pool	NSD	NSD+HiSP
Accuracy	30.4±3.0	20.5±1.0	43.8±4.7	23.7±4.4	55.5±4.1	**87.5±4.6**

4.4 Long-range node regression

Long-Range Ising Model (LRIM). We evaluate inductive node regression on the LRIM benchmark (Mathys et al., 2026), using the `lrim_16_0.6_10k` configuration. Absolute scores on LRIM depend strongly on the sheaf parameterization. To isolate the effect of hierarchy, we fix a diagonal sheaf with stalk dimension 2 and compare NSD with one- and two-stage HiSP with connected selection at pool ratio 0.5 across different depths. The complete results are reported in Appendix D.7.

Two-stage HiSP outperforms NSD at every tested depth. One pooling stage is most effective at moderate depth, whereas two stages continue to improve up to $L = 20$. These results show that, within the tested setting, *depth alone is not a substitute for hierarchy*: increasing depth on the fine graph does not match the benefits of repeated coarsening, which shortens graph distances and makes long-range dependencies accessible to subsequent sheaf-diffusion layers.

Table 4: LRIM-16-hard (Mathys et al., 2026) (test $\log_{10}$MSE ↓).

Model	L=4	L=8	L=12	L=16	L=20
NSD	−0.68±.03	−0.79±.28	−0.61±.14	−0.80±.19	−0.87±.10
HiSP (1 stage)	−0.79±.04	−1.25±.08	−1.27±.28	−0.97±.33	−0.80±.27
HiSP (2 stages)	−0.79±.04	−1.30±.09	−1.43±.06	−1.50±.06	**−1.55±.15**

4.5 Real-world coalition reading: VoteView

We next test whether sheaf-aware hierarchy helps on graphs built from real U.S. Senate communities, with labels taken from observed roll-call coalitions (Appendix D.3).

We construct the dataset from U.S. Senate roll calls in Congresses 90–100 from VoteView (Lewis et al., 2026). Each Congress defines a fixed graph whose nodes represent senators. Within each party, an ideological k-nearest-neighbor graph is augmented with a minimum spanning tree, while dummy corridor nodes connect the two parties across the aisle (Figure 2). Each party is split into two NOMINATE (Poole & Rosenthal, 1985) wings, yielding four communities. Each roll call defines one graph instance. We compute the majority sign within each wing, apply a global sign flip so that the first wing is positive, and retain two coalition patterns: party-line $(+,+,-,-)$ and crossed $(+,-,+,-)$. The resulting dataset contains 1012 balanced instances, with chance accuracy 50%. The labels are derived from observed ballots, but the ballots are not provided as features. For senator i, we use $\mathbf{x}_i = \mathbf{d}_i - \mathbf{m}$, where $\mathbf{d}_i$ is the senator's NOMINATE position and $\mathbf{m}$ is the vote-specific cutting point. Corridor nodes receive the analogous feature based on their layout coordinates. The task is therefore to infer the coalition from the senators' ideological positions relative to the vote-specific cut. We further test sensitivity to local inconsistencies by retaining each senator's feature with probability c and negating it with probability $1-c$. The perturbation is strongest near $c = 0.5$, while c close to zero approaches a coherent global sign reversal; accuracy is therefore not expected to vary monotonically with c. NSD+HiSP achieves the highest mean accuracy at every tested value, while NSD exhibits substantially greater variability. These results, available in Table 5, complement HierBI by supporting a sheaf-aware hierarchy on graph structure derived from real political communities.

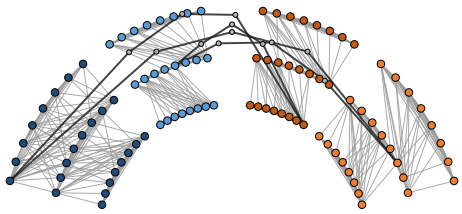


Figure 2: VoteView graph in a hemicycle layout. Blue and orange blocks denote the Democratic and Republican ideological wings, respectively; grey nodes form corridors across the aisle.

Table 5: VoteView test accuracy (%, mean±std; ↑). Features remain two-dimensional; a fraction $1-c$ of senators have their stance negated.

Model	c=1.0	c=0.7	c=0.4	c=0.1
NSD	80.5±3.7	69.9±10.5	73.9±9.4	75.6±13.6
GCN+pool	78.2±2.0	73.6±1.8	73.5±2.0	77.8±1.7
NSD+HiSP	**86.8±1.1**	**78.8±5.4**	**81.6±1.7**	**84.2±1.9**

4.6 Standard benchmarks

Node classification. We insert one or two HiSP stages into NSD backbones to obtain sheaf U-Nets (Appendix C.2). Across nine datasets, hierarchy stays within 1.6 percentage points of unpooled NSD, with the clearest gains on Chameleon and Squirrel and no systematic drop elsewhere (Table 11in Appendix C.2). HiSP is therefore a safe drop-in hierarchical block for node-level sheaf models when a U-Net is desired. Experimental results and details on evaluation protocol are provided in Appendix C.2.

Graph classification. On MUTAG, PROTEINS, ENZYMES, and NCI1 we cross hierarchy with readout (Table 6). The two stages must be compatible: HiSP reduction leaves each supernode in its own root stalk, so a graph-level head that assumes a shared feature space or that was designed for a single fine sheaf can leave the hierarchy unused or actively harmful. The HiSP readout instead summarizes each stalk through descriptors and aggregates those embeddings; it therefore remains meaningful both after coarsening and when applied to unpooled NSD. Depending on the dataset, the readout alone may suffice, both components may help, or coarsening may discard useful fine structure. Details on tuning and evaluation protocol are in Appendix D.8.

4.7 Ablations

We conduct all ablations on HierBI. We study the individual contributions of the HiSP `REDUCE` and `READOUT` components, as well as their interaction. For `REDUCE`, we compare no pooling, direct

Table 6: Graph classification (fold-mean test accuracy, %; ↑).

Model (Readout)	MUTAG	PROTEINS	ENZYMES	NCI1
NSD (SheafPool)	82.0±6.2	65.3±5.3	39.5±7.4	49.9±0.1
NSD (HiSP)	79.9±11.2	69.9±4.2	**41.0±5.7**	**74.6±4.1**
NSD+HiSP (SheafPool)	80.0±11.7	69.8±4.0	26.8±4.7	50.0±0.1
NSD+HiSP (HiSP)	**82.5±5.1**	**74.4±4.1**	37.8±6.4	74.2±3.0

Table 7: HierBI ablations (test accuracy, %; ↑).

Reduce \ Readout	HiSP	mean	SheafPool
no pooling	45.2±4.9	43.5±2.6	47.8±3.0
mean	46.2±2.1	43.4±4.0	47.9±3.2
HiSP	**92.7±1.2**	74.7±19.0	46.9±1.4

mean pooling, and HiSP; for `READOUT`, we compare global mean, SheafPool (Braithwaite et al., 2024), and the HiSP readout. Direct mean pooling provides a coordinate-wise baseline because all node stalks are represented in the same dimension, but it ignores their sheaf-based alignment. Table 7 shows that both HiSP components are required. Without pooling or with direct mean reduction, changing the readout provides no substantial improvement: once the room-level signal is cancelled or left at the fine resolution, the coarse pattern is not reliably recovered. HiSP reduction preserves this signal by aligning features before aggregation, but only the HiSP readout exploits the resulting room representations consistently. The complete configuration achieves 92.7%, confirming the complementary roles of `REDUCE` and `READOUT`.

4.8 Implementation

The sheaf layer in every experiment is a PyTorch Geometric implementation of neural sheaf diffusion. Appendix E compares it with the original code on the released checkpoints: converted weights reproduce the original logits to numerical precision. Training runs in `tgp`. For pooling operators with fixed assignments and connectivity, its `PreCoarsening` transform constructs and stores the graph hierarchy once, while `PoolDataLoader` collates the precomputed levels into mini-batches. We extend this pipeline to cache the index structure of each sheaf Laplacian and the rooted cluster trees required by HiSP. During batching, sheaf indices are shifted by the corresponding numbers of edges, nodes, or stalk coordinates, while tree parents are stored as relative node indices. All structural fields are therefore computed once per graph rather than reconstructed at every epoch. Appendix F describes the cached fields and collation rules.

5 Limitations and Conclusion

Changing the root of a fixed transport tree reverses the edges on the path between the old and new roots. Adjoint transport generally differs from inverse transport even for invertible maps. With orthogonal restriction maps, the regularized transport is $\tau_{u\to v} = (1+\varepsilon)^{-1}\rho^*_{e,v}\rho_{e,u}$, so a path of length m introduces attenuation $(1+\varepsilon)^{-m}$, which can contribute to root dependence. How the root should be found (e.g. by sampling, by a learned selection, or by some other criterion) is a separate question from the construction of the transport, and we leave it open.

This work introduces HiSP together with the implementation required for hierarchical sheaf learning: a PyG sheaf convolution, a pooling layer following the SRC interface, and a sheaf-aware readout. These pieces are what allow a sheaf network to be trained on a graph-level task, coarsening the sheaf and learning a new one at each resolution, and what allow the same encoder to be lifted back to the nodes as a sheaf U-Net. The experiments then use that architecture to show the behaviour the operator is built for. Each resolution can learn a sheaf independent of the one below it, so a coarse compatibility constraint need not be visible on the fine edges. Transport inside a cluster can keep a task-relevant stalk and discard the rest. Coarsening can bring distant nodes into a common receptive field within a fixed layer budget.

Reproducibility Statement

Code for reproducing the experiments and results is available at https://anonymous.4open.science/r/hisp/ and will be released publicly upon publication. We use both publicly available datasets and synthetic datasets developed for this study. The synthetic-data generators are released along with the rest of the code, and the paper provides the task definitions and all experimental details needed to reproduce the results.

AI Use Statement

Generative AI tools assisted with code development, mathematical derivations, initial drafting of the appendix, and targeted edits in the main text. The authors checked all AI-assisted content, calculations, and citations and take full responsibility for the content of the paper.

References

Carlo Abate and Filippo Maria Bianchi. Maxcutpool: differentiable feature-aware maxcut for pooling in graph neural networks, 2025. URL https://openreview.net/forum?id=xlbXRJ2XCP.

Carlo Abate, Ivan Marisca, and Filippo Maria Bianchi. Torch geometric pool: the pytorch library for pooling in graph neural networks, 2026. URL https://arxiv.org/abs/2512.12642.

Davide Bacciu, Alessio Conte, and Francesco Landolfi. Generalizing downsampling from regular data to graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 6718–6727, 2023.

Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, Michael Bronstein, Petar Veličković, and Pietro Liò. Sheaf neural networks with connection laplacians. In *Topological, Algebraic and Geometric Learning Workshops 2022*, pp. 28–36. PMLR, 2022a.

Federico Barbero, Cristian Bodnar, Haitz Sáez de Ocáriz Borde, and Pietro Lio. Sheaf attention networks. In *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*, 2022b.

Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Mincut pooling in graph neural networks. *arXiv preprint arXiv:1907.00481*, 5, 2019.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F Montufar, and Michael Bronstein. Weisfeiler and Lehman go cellular: CW networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 2625–2640, 2021.

Cristian Bodnar, Francesco Di Giovanni, Benjamin Chamberlain, Pietro Lio, and Michael Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. *Advances in Neural Information Processing Systems*, 35:18527–18541, 2022.

Alessio Borgi, Fabrizio Silvestri, and Pietro Liò. Polynomial neural sheaf diffusion: A spectral filtering approach on cellular sheaves. *arXiv preprint arXiv:2512.00242*, 2025.

Luke Braithwaite, Alessio Borgi, Gabriele Onorato, Kristjan Tarantelli, Francesco Restuccia, Fabrizio Silvestri, and Pietro Liò. Heterogeneous sheaf neural networks. *arXiv preprint arXiv:2409.08036*, 2024.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE transactions on neural networks and learning systems*, 35(2):2708–2718, 2022.

Jakob Hansen and Thomas Gebhart. Sheaf neural networks. *arXiv preprint arXiv:2012.06333*, 2020.

Jakob Hansen and Robert Ghrist. Toward a spectral theory of cellular sheaves. *Journal of Applied and Computational Topology*, 3(4):315–358, 2019.

Jakob Hansen and Robert Ghrist. Opinion dynamics on discourse sheaves. *SIAM Journal on Applied Mathematics*, 81(5):2033–2060, 2021.

Per Christian Hansen. *Rank-deficient and discrete ill-posed problems: numerical aspects of linear inversion*. SIAM, 1998.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Jeffrey B. Lewis, Keith Poole, Howard Rosenthal, Adam Boche, Aaron Rudkin, and Luke Sonnet. Voteview: Congressional roll-call votes database. https://voteview.com/, 2026. Accessed 2026.

Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 723–731, 2019.

Joël Mathys, Henrik Christiansen, Federico Errica, Takashi Maruyama, and Francesco Alesiani. LRIM: A physics-based benchmark for provably evaluating long-range capabilities in graph learning. In *International Conference on Learning Representations*, 2026. URL https://mlanthology.org/iclr/2026/mathys2026iclr-lrim/.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. TUDataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020. URL https://www.graphlearning.io.

Keith T. Poole and Howard Rosenthal. A spatial model for legislative roll call analysis. *American Journal of Political Science*, 29(2):357–384, 1985.

Alex Pothen, Horst D Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM journal on matrix analysis and applications*, 11(3):430–452, 1990.

Antonio Purificato, Giulia Cassarà, Federico Siciliano, Pietro Liò, and Fabrizio Silvestri. Sheaf4rec: Sheaf neural networks for graph-based recommender systems. *ACM Transactions on Recommender Systems*, 4(2):1–26, 2025.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Olga Zaghen, Antonio Longa, Steve Azzolin, Lev Telyatnikov, Andrea Passerini, and Pietro Lio. Sheaf diffusion goes nonlinear: Enhancing gnns with adaptive sheaf laplacians. In *ICML 2024 Workshop on Geometry-grounded Representation Learning and Generative Modeling*, 2024.

APPENDIX

A SELECTION STRATEGIES

Both selection strategies depend only on the topology of the input graph. Their cluster assignments and representatives can therefore be computed once before training and reused at every forward pass.

k-MIS. Given an integer k, k-MIS constructs a maximal k-independent set $R \subseteq V$. Its representatives are mutually separated,

$$\mathrm{dist}_G(r_i, r_j) > k \qquad \text{for all distinct } r_i, r_j \in R,$$

while maximality ensures that every vertex lies within k hops of at least one representative. Vertices are assigned to nearby representatives through a multi-source graph traversal, with ties resolved deterministically. This produces a hard partition $\mathcal{C} = \{C_1, \dots, C_K\}$, where each cluster C_a contains its representative r_a and is connected.

Recursive spectral selection. Starting from each connected component of G, recursive spectral selection computes the Fiedler vector of the Laplacian of the current induced subgraph and bisects its vertices at the median Fiedler coordinate. If either side is disconnected, it is separated into its connected components before recursion continues. The procedure recursively splits the largest remaining connected subsets until the target number of clusters is reached. For each final cluster C_a, we select

$$r_a \in \arg\max_{v \in C_a} \deg_{G[C_a]}(v)$$

as its representative, breaking ties deterministically.

Output. Both procedures return a hard assignment matrix

$$\mathbf{S} \in \{0,1\}^{|V| \times K}, \qquad C_a = \{v \in V : S_{va} = 1\},$$

whose columns induce K connected clusters, together with a root $r_a \in C_a$ for each cluster. The roots initialize the transport trees used by `REDUCE`.

B REGULARIZED AND NON-EXPANSIVE TRANSPORT MAPS

This section provides further details on the transport maps introduced in Section 3.3.1. We derive the edgewise transport from a regularized least-squares problem, explain why regularization alone does not prevent pathwise amplification, and establish the stability of the normalized transport used by HiSP. As in the main text, we omit the layer index ℓ.

Edgewise transport. Consider a cluster C_a with root r_a, and let T_a be a spanning tree of the induced subgraph $G[C_a]$. Rooting T_a orients every tree edge from parent to child. For an edge $e = \{u, v\}$, let $u = p_a(v)$ be the parent of v. The restriction maps

$$\rho_{e,u} : \mathcal{F}(u) \longrightarrow \mathcal{F}(e), \qquad \rho_{e,v} : \mathcal{F}(v) \longrightarrow \mathcal{F}(e)$$

map the endpoint stalks to the common edge stalk.

Given $\mathbf{x}_u \in \mathcal{F}(u)$, its representation in $\mathcal{F}(e)$ is $\rho_{e,u}\mathbf{x}_u$. To obtain a corresponding vector in $\mathcal{F}(v)$, we solve

$$\mathbf{y}^\star = \arg\min_{\mathbf{y} \in \mathcal{F}(v)} \|\rho_{e,v}\mathbf{y} - \rho_{e,u}\mathbf{x}_u\|_2^2 + \varepsilon\|\mathbf{y}\|_2^2, \qquad \varepsilon > 0. \tag{12}$$

The first term measures disagreement in the edge stalk, while the second penalizes solutions with large norm. The corresponding normal equations are

$$\left(\rho_{e,v}^*\rho_{e,v} + \varepsilon\mathbf{I}_d\right)\mathbf{y}^\star = \rho_{e,v}^*\rho_{e,u}\mathbf{x}_u.$$

Since $\varepsilon > 0$, the matrix on the left-hand side is positive definite and therefore invertible. Consequently,

$$\mathbf{y}^\star = \left(\rho_{e,v}^*\rho_{e,v} + \varepsilon\mathbf{I}_d\right)^{-1}\rho_{e,v}^*\rho_{e,u}\mathbf{x}_u.$$

This defines the Tikhonov-regularized transport (Hansen, 1998)

$$\tau_{u\to v} = \left(\rho_{e,v}^*\rho_{e,v} + \varepsilon\mathbf{I}_d\right)^{-1}\rho_{e,v}^*\rho_{e,u} : \mathcal{F}(u) \longrightarrow \mathcal{F}(v). \tag{13}$$

Regularization does not imply non-expansiveness. The regularized inverse

$$\rho_{e,v}^{+,\varepsilon} = \left(\rho_{e,v}^{*}\rho_{e,v} + \varepsilon \mathbf{I}_d\right)^{-1} \rho_{e,v}^{*}$$

is well defined even when $\rho_{e,v}$ is singular. If σ is a singular value of $\rho_{e,v}$, the corresponding singular value of $\rho_{e,v}^{+,\varepsilon}$ is

$$g_\varepsilon(\sigma) = \frac{\sigma}{\sigma^2 + \varepsilon}.$$

Since

$$\max_{\sigma \geq 0} g_\varepsilon(\sigma) = \frac{1}{2\sqrt{\varepsilon}},$$

the regularization bounds the amplification due to the inverse, but this bound may still be greater than one. In particular,

$$\|\tau_{u\to v}\|_2 \leq \frac{\|\rho_{e,u}\|_2}{2\sqrt{\varepsilon}},$$

which does not generally imply $\|\tau_{u\to v}\|_2 \leq 1$.

This distinction becomes important when transports are composed. Let

$$\pi_{r_a \to v} = (v_0 = r_a, v_1, \ldots, v_m = v)$$

be the unique path from r_a to v in T_a. Without normalization, the corresponding root-to-node transport is

$$F_{r_a\to v} = \tau_{v_{m-1}\to v_m} \cdots \tau_{v_0 \to v_1}.$$

By submultiplicativity of the operator norm,

$$\|F_{r_a\to v}\|_2 \leq \prod_{j=1}^{m} \|\tau_{v_{j-1}\to v_j}\|_2. \tag{14}$$

Thus, if every edge transport has norm at most $1+\delta$, the worst-case bound grows as $(1+\delta)^m$. Even moderate edgewise amplification may therefore accumulate along long paths, making the scale of the pooled features depend strongly on the cluster diameter and potentially destabilizing activations and gradients.

Non-expansive normalization. To prevent pathwise amplification, we rescale every edge transport as

$$\widehat{\tau}_{u\to v} = \frac{\tau_{u\to v}}{\max\{1, \|\tau_{u\to v}\|_2\}}, \qquad \|\widehat{\tau}_{u\to v}\|_2 \leq 1. \tag{15}$$

This operation leaves $\tau_{u\to v}$ unchanged when it is already non-expansive and rescales it otherwise. Because the entire map is multiplied by a scalar, the normalization preserves its rank, singular vectors, and relative singular values. It therefore retains the anisotropic structure encoded by the learned restriction maps.

This normalization only rescales the transport; it does not make it norm-preserving. Directions that were attenuated remain attenuated, and the kernel of the transport is unchanged. It therefore prevents amplification without recovering information already discarded by the transport.

Composition along the rooted tree. The normalized root-to-node transport

$$F_{r_a\to v} : \mathcal{F}(r_a) \longrightarrow \mathcal{F}(v)$$

is defined recursively by

$$F_{r_a\to r_a} = \mathbf{I}_d, \qquad F_{r_a\to v} = \widehat{\tau}_{p_a(v)\to v} F_{r_a \to p_a(v)}. \tag{16}$$

Because T_a contains a unique path from r_a to every $v \in C_a$, this recursion is well defined. Moreover,

$$\|F_{r_a\to v}\|_2 \leq \prod_{(u\to w)\in \pi_{r_a\to v}} \|\widehat{\tau}_{u\to w}\|_2 \leq 1. \tag{17}$$

Hence, the complete root-to-node transport is non-expansive. Since the operator norm is invariant under adjunction,

$$\|F_{r_a\to v}^{*}\|_2 = \|F_{r_a\to v}\|_2 \leq 1.$$

It follows that, for vectors $\mathbf{z} \in \mathcal{F}(r_a)$ and $\mathbf{x}_v \in \mathcal{F}(v)$,

$$\|F_{r_a \to v}\mathbf{z}\|_2 \le \|\mathbf{z}\|_2, \qquad \|F^*_{r_a \to v}\mathbf{x}_v\|_2 \le \|\mathbf{x}_v\|_2. \tag{18}$$

For matrix-valued node features $\mathbf{X}_v \in \mathbb{R}^{d\times h}$, the analogous Frobenius-norm bound is

$$\|F^*_{r_a \to v}\mathbf{X}_v\|_{\mathrm{F}} \le \|\mathbf{X}_v\|_{\mathrm{F}}.$$

The same stability guarantee therefore applies both to the root-to-node transport used for lifting and to the aligned feature

$$\mathbf{Z}_{a,v} = F^*_{r_a \to v}\mathbf{X}_v$$

used for pooling.

Attenuation and path dependence. Non-expansiveness prevents exploding feature norms, but products of transports with norm strictly below one may attenuate features along long paths. This motivates selecting compact clusters, since shorter root-to-node paths reduce both amplification before normalization and attenuation after normalization.

The construction also depends on the selected root and spanning tree. For a tree edge $e = \{u, v\}$, the reverse regularized transport is

$$\tau_{v\to u} = \left(\rho^*_{e,u}\rho_{e,u} + \varepsilon\mathbf{I}_d\right)^{-1}\rho^*_{e,u}\rho_{e,v}.$$

In general,

$$\tau_{v\to u} \ne \tau^{-1}_{u\to v} \qquad \text{and} \qquad \tau_{v\to u} \ne \tau^*_{u\to v}.$$

Therefore, applying $F^*_{r_a\to v}$ to align a node feature with the root is not equivalent to composing independently computed reverse least-squares transports. Changing the root may reverse some edge orientations, while changing the spanning tree may select different paths, leading to a different pooled representation. Sampling roots and spanning trees can mitigate this dependence by exposing the model to multiple orientations and transport paths.

Stochastic tree resampling. To reduce sensitivity to a single path system, we may keep the clusters and roots fixed while resampling the BFS spanning trees at each forward pass. Let $\mathcal{P}$ denote the resulting parameter-independent distribution over tree collections $T = \{T_a\}_{a=1}^K$. Training then minimizes the expected loss

$$J(\theta) = \mathbb{E}_{T\sim\mathcal{P}}\left[\mathcal{L}\big(\widehat{\mathbf{y}}(T;\theta), \mathbf{y}\big)\right].$$

Under the usual conditions for interchanging differentiation and expectation, a tree sampled at each forward pass gives an unbiased stochastic gradient:

$$\mathbb{E}_{T\sim\mathcal{P}}\left[\nabla_\theta\mathcal{L}\big(\widehat{\mathbf{y}}(T;\theta), \mathbf{y}\big)\right] = \nabla_\theta J(\theta).$$

At evaluation, the pooled representation can be averaged over M independent tree samples:

$$\overline{\mathbf{X}}^{(M)}_c = \frac{1}{M}\sum_{m=1}^{M}\mathbf{X}_c\Big(T^{(m)}\Big), \qquad T^{(m)} \overset{\text{i.i.d.}}{\sim} \mathcal{P}.$$

This estimator approximates $\mathbb{E}_T[\mathbf{X}_c(T)]$ as M increases. Because the downstream predictor is nonlinear, applying it to the averaged representation is not, in general, equivalent to averaging the predictions obtained from individual trees. Tree resampling therefore marginalizes over path choices rather than making the transport path-independent.

C Hierarchical Sheaf Neural Network Architectures

We refer to an architecture that alternates sheaf convolution and HiSP layers as a HSNN. At hierarchy level ℓ, a sheaf-convolution block with parameters θ_ℓ learns the restriction maps $\boldsymbol{\rho}^{(\ell)}$ and updates the node features to $\widetilde{\mathbf{X}}^{(\ell)}$. The subsequent HiSP layer uses these restriction maps to align and aggregate the updated features:

$$\big(G_\ell, \mathbf{X}^{(\ell)}\big) \xrightarrow{\mathrm{SheafConv}_{\theta_\ell}} \big(G_\ell, \boldsymbol{\rho}^{(\ell)}, \widetilde{\mathbf{X}}^{(\ell)}\big) \xrightarrow{\mathrm{HiSP}_\ell} \big(G_{\ell+1}, \mathbf{X}^{(\ell+1)}\big). \tag{19}$$

The subsequent sheaf-convolution block learns a new sheaf on $G_{\ell+1}$, allowing each hierarchy level to learn diffusion dynamics adapted to its graph resolution.

C.1 GRAPH-LEVEL ARCHITECTURE

For graph-level tasks, the model repeatedly applies the hierarchical stage in Equation (19). Denoting the composition of the sheaf-convolution and pooling operations at level ℓ by

$$\mathcal{H}_\ell = \mathrm{HiSP}_\ell \circ \mathrm{SheafConv}_{\theta_\ell},$$

the complete architecture is

$$\left(G_0, \mathbf{X}^{(0)}\right) \xrightarrow{\mathcal{H}_0} \left(G_1, \mathbf{X}^{(1)}\right) \xrightarrow{\mathcal{H}_1} \cdots \xrightarrow{\mathcal{H}_{L-1}} \left(G_L, \mathbf{X}^{(L)}\right) \xrightarrow{\text{HiSPReadout}} \mathbf{g}_G \xrightarrow{\text{MLP}} \widehat{\mathbf{y}}_G. \tag{20}$$

The hierarchy progressively replaces groups of nodes with supernodes. The HiSP readout maps the final coarse representations to a graph descriptor $\mathbf{g}_G$, and the prediction head returns one output $\widehat{\mathbf{y}}_G$ per graph. No lifting or decoder is required.

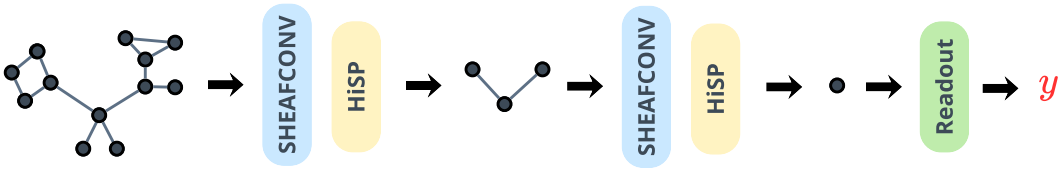


Figure 3: Graph-level HSNN. Each encoder stage applies a sheaf-convolution block followed by HiSP pooling. A HiSP readout aggregates the representation at the coarsest resolution to produce the graph-level prediction.

C.2 NODE-LEVEL ARCHITECTURE

For node-level tasks, the hierarchical encoder is complemented by a decoder. At encoder level ℓ, the output of the sheaf-convolution block is stored as a skip representation:

$$\mathbf{K}^{(\ell)} = \mathrm{SheafConv}_{\theta_\ell^{\downarrow}}\left(G_\ell, \mathbf{X}^{(\ell)}\right), \tag{21}$$

$$\left(G_{\ell+1}, \mathbf{X}^{(\ell+1)}\right) = \mathrm{HiSP}_\ell\left(G_\ell, \mathbf{K}^{(\ell)}; \boldsymbol{\rho}^{(\ell)}, \pi_\ell\right). \tag{22}$$

The restriction maps, cluster assignments, and transport frames computed during pooling are retained for the corresponding decoder stage.

Let $\pi_\ell : V_\ell \to V_{\ell+1}$ denote the cluster assignment, and suppose that $\pi_\ell(v) = r_a$. The lifted feature at v is

$$\mathbf{U}_v^{(\ell)} = F_{r_a \to v}^{(\ell)} \mathbf{D}_{r_a}^{(\ell+1)}. \tag{23}$$

The lifted representation is combined with the encoder skip connection and refined by a decoder sheaf-convolution block:

$$\mathbf{D}^{(\ell)} = \mathrm{SheafConv}_{\theta_\ell^{\uparrow}}\left(G_\ell, \mathbf{U}^{(\ell)} \oplus \mathbf{K}^{(\ell)}\right), \tag{24}$$

where $\oplus$ denotes either addition or projected concatenation. After the last decoder stage, a node-wise prediction head produces

$$\widehat{\mathbf{y}}_v = \mathrm{MLP}\left(\mathbf{D}_v^{(0)}\right), \qquad v \in V_0. \tag{25}$$

Thus, the encoder learns coarse representations, while the decoder restores the original resolution using the same transport geometry employed during pooling.

D EXPERIMENTAL DETAILS

This appendix collects dataset constructions, training protocols, and extended tables that support Section 4. Unless stated otherwise, HiSP uses connected cluster selection, *random BFS* tree sampling for transport (Section 3.3), and weighted transport-mean aggregation.

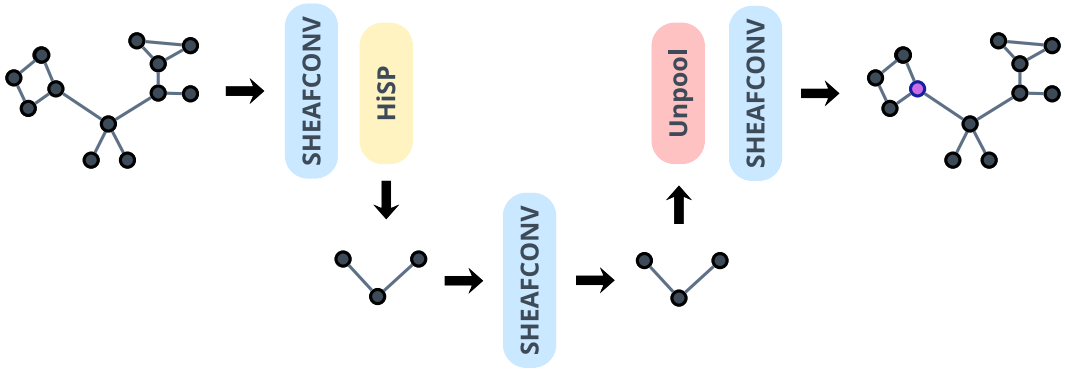


Figure 4: Node-level HSNN. HiSP constructs the coarse encoder representation. The decoder lifts it to the original graph, combines it with the corresponding skip connection, and applies a sheaf-convolution block before producing node-level predictions.

D.1 Dataset statistics

Table 8–10 summarize the datasets used in this paper (style following Torch Geometric Pool (Abate et al., 2026)). Synthetic sizes are measured from the generators used in our recipes; geom-gcn and TUDataset sizes match the training loaders.

Table 8: Synthetic / semi-synthetic benchmarks.

Dataset	# Graphs	Avg. # Nodes	Avg. # Edges	# Feat.	# Classes
HierBI	800	76.0	272.0	4	4
VoteView	1012	96.8	866.7	2	2
RingTransfer	1200	24.0	24.0	5	5
Marked PathCopy	1200	31.5	30.5	8	5

Table 9: Heterophilic node-classification datasets (geom-gcn). Homophily values $h(\mathcal{G})$ from Bodnar et al. (2022).

Dataset	# Nodes	# Edges	# Feat.	# Classes	$h(\mathcal{G})$
Texas	183	279	1703	5	0.11
Wisconsin	251	450	1703	5	0.21
Cornell	183	277	1703	5	0.30
Chameleon	2277	18050	2325	5	0.23
Squirrel	5201	108536	2089	5	0.22

D.2 HierBI

Four rooms are copies of $K_{8,8}$ (64 room nodes). Rooms $0, 1$ form a meta-left pair and rooms $2, 3$ a meta-right pair. Every left–right pair is joined by a corridor of length three, so after pooling one supernode per room the coarse graph is $K_{2,2}$. Corridor dummies bring each graph to 76 nodes and 272 edges (Figure 5). Features are four-dimensional, the direct sum of two subspaces. A hidden sign pattern frames each subspace so that an unsigned room mean cancels. The label is $y = 2b_{\text{coarse}} + b_{\text{fine}}$, the pair of opposite-versus-crossed bits, and node features include isotropic noise with σ=0.5. Corridor nodes are zero.

Training uses 400/200/200 splits, hidden width 16, K=4 connected rooms, and random BFS transport trees, for at most 100 epochs. HierBI uses a diagonal sheaf with d=2, one pre-pool and two post-pool sheaf layers, Adam with lr=0.01, and patience 80. layer, Adam with lr=0.05, and patience 30. The reduce and readout comparison is Table 7.

Table 10: TUDataset graph-classification benchmarks (Morris et al., 2020).

Dataset	# Graphs	Avg. # Nodes	Avg. # Edges	# Feat.	# Classes
MUTAG	188	17.9	19.8	7	2
PROTEINS	1113	39.1	72.8	3	2
ENZYMES	600	32.6	62.1	3	6
NCI1	4110	29.9	32.3	37	2

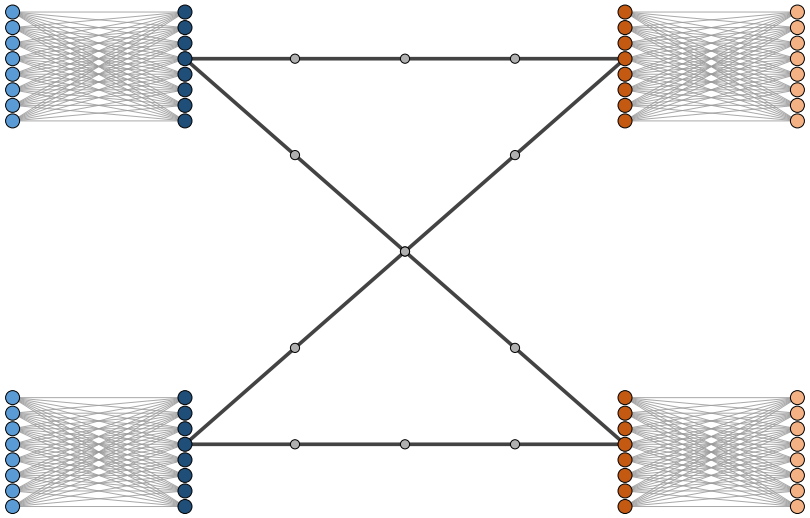


Figure 5: HierBI sample. Colours mark the four rooms; grey nodes are corridor dummies that connect every left room to every right room.

D.3 VoteView coalition reading

We build graphs from U.S. Senate members and roll calls in Congresses 90–100 (less-polarized era in which both party-line and crossed wing coalitions appear in real votes). Each Congress has a fixed ideology skeleton: senators are placed by NOMINATE (Poole & Rosenthal, 1985) coordinates, each party is split into two wings, senators within a wing are linked by ideology kNN (k=8) plus an MST, and Democrat wings meet Republican wings only through corridor dummies (Figure 2). Each roll call is one instance. Labels are the observed wing-majority patterns A $(+,+,-,-)$ and B $(+,-,+,-)$ after a global sign flip that makes the first wing positive; ballots themselves are not features. Features are relative NOMINATE positions $\mathbf{x}_i = \mathbf{d}_i - \mathbf{m}$ with respect to the vote-specific cut $\mathbf{m}$. Balancing the two classes yields 1012 graphs (mean 96.8 nodes, 866.7 edges). The coherence sweep of Table 5 retains each senator's feature with probability c and negates it otherwise.

Training uses a stratified $70/15/15$ split, batch size 32, hidden width 16, K=4 connected rooms, random BFS transport trees, a diagonal sheaf with d=1, one pre-pool and one post-pool layer, Adam with lr=0.01, at most 100 epochs, and patience 30 on validation accuracy.

D.4 Marked PathCopy

Each graph is a path of length 24 with a small random padding (about 32–35 nodes). Source and query sit at the path endpoints. Node features mark the source, query, and distractors, and carry a five-way one-hot value channel. Wrong-class distractors occupy every second interior node (Figure 6). The model is supervised only at the query. We use two connected HiSP stages at ratio 0.5 with random BFS transport trees, so that after both collapses source and query remain in distinct coarse rooms. Training uses $800/200/200$ graphs, 16 sheaf layers in a U-Net split, hidden width 64,

diagonal sheaf with d=2, Adam with lr=0.02, cosine warmup of 10 epochs, at most 150 epochs, and patience 50. Baselines share this schedule.



Figure 6: Marked PathCopy sample. Endpoints are the source (left) and query (right); every second interior node carries a wrong-class distractor.

D.5 RingTransfer

We follow Bodnar et al. (2021): chordless cycles of 24 nodes, five classes, query at node 0 and source at the antipode (distance 12). Features are constant ones except for a wiped query and a one-hot source. Models are masked node classifiers. NSD+HiSP uses one connected stage with K=6 rooms, random BFS transport trees, diagonal sheaf with d=3, and hidden width 64. Train/val/test sizes are 800/200/200. Because the diameter equals half the cycle length, any fixed-depth message passer whose receptive field is shorter than 12 hops is information-theoretically unable to solve the task; coarsening is what reduces that diameter.

D.6 Node Classification

We follow the evaluation and model-selection protocol of Bodnar et al. (2022). All models are evaluated on the same ten fixed Geom-GCN splits, each containing 48%, 32%, and 20% of the nodes from every class for training, validation, and testing, respectively. For each split and hyperparameter configuration, training uses Adam and the checkpoint with the highest validation accuracy is retained. Following the discrete NSD protocol, training is limited to 1000 epochs with patience 100 on Chameleon and Squirrel, and to 500 epochs with patience 200 on the remaining datasets.

We run separate hyperparameter sweeps for the one- and two-stage sheaf U-Net architectures. Each sweep configuration is evaluated on all ten splits, and the configuration with the highest mean validation accuracy is selected. We report the mean and standard deviation of the test accuracies obtained at the corresponding validation-selected checkpoints. When reporting a single NSD+HiSP result, we similarly select between the one- and two-stage configurations using mean validation accuracy. The complete per-stage results are reported in Table 11.

Table 11: Node classification (test acc., %). NSD uses our reproduction on Texas, Wisconsin, Cornell, Chameleon, and Squirrel (released recipes); Film, CiteSeer, PubMed, and Cora quote the best published NSD variant of Bodnar et al. (2022). HiSP: one- and two-stage sheaf U-Nets. H2GCN / GCN from Bodnar et al. (2022, Table 1).

Dataset	NSD	HiSP (1)	HiSP (2)	H2GCN	GCN
Film	**37.81±1.15**	37.0±1.2	36.9±1.2	35.70±1.00	27.32±1.10
PubMed	89.49±0.40	89.5±0.4	**89.7±0.3**	89.49±0.38	88.42±0.50
CiteSeer	77.14±1.85	76.6±1.4	**77.4±1.4**	77.11±1.57	76.50±1.36
Cora	87.30±1.15	87.3±1.2	87.2±1.0	**87.87±1.20**	86.98±1.27
Squirrel	57.00±1.90	**58.0±2.1**	57.1±1.8	36.48±1.86	53.43±2.01
Chameleon	68.00±1.22	69.0±1.9	**69.3±1.9**	60.11±2.15	64.82±2.24
Texas	**85.14±5.30**	81.4±5.1	85.1±5.8	84.86±7.23	55.14±5.16
Cornell	**83.24±6.37**	83.0±5.1	82.4±6.8	82.70±5.28	60.54±5.30
Wisconsin	**89.41±2.93**	87.5±3.5	87.8±1.7	87.65±4.98	51.76±3.06

D.7 LRIM protocol and full diag_d2 grid

Protocol. We use the official inductive split of `lrim_16_0.6_10k` (Mathys et al., 2026) (80/10/10), metric $\log_{10}$(MSE). All models in this section share the `diag_d2` sheaf (diagonal restriction maps, stalk dimension 2). We sweep depth $L \in \{4, 8, 12, 16, 20\}$ and compare unpooled NSD to HiSP with one or two pooling stages, using either connected selection at ratios $\{0.2, 0.5, 0.8\}$ or κ-MIS selection with $\kappa \in \{1, 2, 3, 4\}$. Layer budgets are matched by splitting sheaf layers across encoder/decoder stages. Each cell is mean ± std over five seeds.

Why `diag_d2`. Richer sheaves (e.g. general maps with high/low-pass residuals) change absolute LRIM scores substantially and can dominate the comparison. Fixing a simple sheaf makes the depth×pooling trend readable: the question here is whether hierarchical coarsening helps under a matched sheaf, not which sheaf wins the leaderboard.

Extended results. Table 12 expands Table 4. Two-stage pooling is consistently stronger than NSD; one-stage pooling peaks at moderate L and then degrades; k-MIS follows the same qualitative pattern as connected selection.

Table 12: LRIM-16-hard, `diag_d2` only (test $\log_{10}$MSE ↓).

Model	L=4	L=8	L=12	L=16	L=20
depth-L cutoff	−0.287	−2.066	−13.782	−13.782	−13.782
NSD	−0.682±.026	−0.793±.278	−0.607±.135	−0.804±.185	−0.865±.098
One stage, connected					
r=0.2	−0.964±.035	−1.358±.052	−1.289±.224	−0.688±.252	−0.691±.304
r=0.5	−0.787±.043	−1.250±.082	−1.267±.283	−0.973±.333	−0.799±.268
r=0.8	−0.684±.010	−1.016±.360	−1.022±.238	−0.911±.304	−0.711±.208
One stage, k-MIS					
κ=1	−0.823±.042	−1.270±.114	−1.170±.226	−1.036±.245	−0.841±.293
κ=2	−1.023±.041	−1.419±.096	−1.290±.276	−0.916±.392	−0.904±.267
κ=3	−1.069±.047	−1.372±.074	−1.197±.344	−0.741±.182	−0.585±.329
κ=4	−1.058±.065	−1.350±.076	−0.991±.420	−0.443±.393	−0.483±.255
Two stages, connected					
r=0.2	−0.899±.085	−1.356±.043	−1.389±.047	−1.465±.106	−1.372±.136
r=0.5	−0.787±.041	−1.299±.093	−1.433±.062	−1.500±.060	−1.554±.151
r=0.8	−0.673±.010	−1.301±.052	−1.285±.105	−1.376±.226	−1.485±.188
Two stages, k-MIS					
κ=1	−0.893±.029	−1.314±.104	−1.491±.023	−1.441±.139	−1.374±.232
κ=2	−0.997±.055	−1.425±.049	−1.438±.100	−1.466±.152	−1.253±.409
κ=3	−0.946±.099	−1.429±.022	−1.409±.046	−1.454±.086	−1.126±.281
κ=4	−0.875±.063	−1.372±.060	−1.368±.086	−1.506±.063	−0.987±.192

D.8 Graph-level sweep protocol

We run Bayesian sweeps on MUTAG, PROTEINS, ENZYMES, and NCI1. Each trial trains a 10-fold stratified split and logs mean validation loss. Search includes sheaf type/dimension (diagonal, bundle, general; with or without high/low-pass residuals), depth $L \in [2, 8]$, hidden width in $\{8, 16, 32, 64\}$, dropout, input dropout, and sheaf/weight decay. Pooled variants additionally sweep the number of HiSP stages ($\{1, 2\}$) with k-MIS (k=1), random BFS trees, and weighted aggregation.

E NSD port and Sheaf U-Net

E.1 NSD reproduction and HiSP comparison

On the five geom-gcn datasets with public recipes (Texas, Wisconsin, Cornell, Chameleon, Squirrel), we reproduce unpooled NSD from the released checkpoints of Bodnar et al. (2022) (diagonal or $O(d)$-bundle sheaves; seed 43). Our PyG/tgp port matches the original logits on all fifty folds (max $|\Delta\text{logit}| \approx 3\times10^{-5}$). For Film, CiteSeer, PubMed, and Cora the recipes were not released, so the NSD column quotes the best published variant from their Table 1. Table 11 reports that backbone against one- and two-stage HiSP sheaf U-Nets under the same Geom-GCN splits and validation protocol (Appendix D.6), with H2GCN and GCN included for context.

E.2 SHEAF U-NET DETAILS

Node-level HSNNs embed the hierarchical encoder in a sheaf U-Net with transport-based lifting (Appendix C.2). Checkpoints are selected by validation accuracy over ten folds; the corresponding test accuracies are Table 11.

E.3 SHEAFCONV PORT TIMING

To check that the PyG/tgp sheaf-convolution port is faithful, we time eval forwards on several node datasets and a MUTAG batch (CUDA, RTX 3080) with and without a cached Laplacian / pooling pretransform (Figure 7). NSD (original) and NSD (PyG) match under both settings; caching the pretransform yields about a $4\times$ speedup. The same pattern holds for a one-stage sheaf U-Net and for NSD+HiSP on MUTAG.

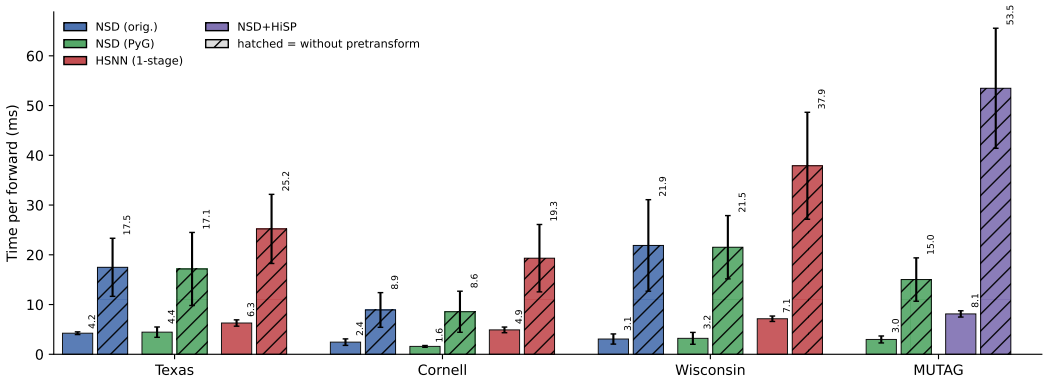


Figure 7: Eval forward time with vs. without sheaf/pool pretransform on Texas, Cornell, Wisconsin, and MUTAG. NSD (original) and NSD (PyG) match; pretransform caching removes a large constant cost.

F CACHED SHEAF INDICES AND HISP TREES

`tgp` stores a precomputed pooling assignment on each graph and batches those assignments with its own collation. A sheaf network needs two further pieces, both fixed once the clusters and the graph topology are fixed: the index structure of the sheaf Laplacian, and the rooted tree used by HiSP. `SheafNetPretransform` writes them into two lists on the same `Data` object.

```
SheafNetPretransform(G):
    sheaf_data[0] <- LaplacianIndices(G)      # level 0
    for level l = 0, ..., L-1:
        pooled_data[l] <- HiSPSelect(G_l)     # clusters, roots, parents
        G_{l+1}        <- coarse graph of pooled_data[l]
        sheaf_data[l+1] <- LaplacianIndices(G_{l+1})
```

Listing 1: Cached sheaf / HiSP pretransform.

`LaplacianIndices` is a `SheafData` object. Its fields are integer indices into edges, into nodes, or into the lifted stalks of size $|V| \cdot d$.`HiSPSelect` stores, for each cluster, the assignment and the parent of every node in the transport tree. Parents are kept as a difference from the node index, so concatenation does not require a second offset rule. Batching uses those increments explicitly:

```
SheafData.__inc__(key):
    edge index   -> |E|
    node index   -> |V|
    lifted index -> |V| * d
    tree parent  -> 0          # already stored as parent - node
```

Listing 2: Batch index increments for sheaf fields.

`SheafNetBatch` collates ordinary PyG fields, the `pooled_data` list, and the `sheaf_data` list in one step. At training time the sheaf layer reads `sheaf_data[l]` and HiSP reads the tree in `pooled_data[l]`, without rebuilding either.