Grothendieck Graph Neural Networks Framework: An Algebraic Platform for Crafting Topology-Aware GNNs

Anonymous authors
Paper under double-blind review

Abstract

Graph Neural Networks (GNNs) are almost universally built on a single primitive: the neighborhood. Regardless of architectural variations, message passing ultimately aggregates over neighborhoods, which intrinsically limits expressivity and often yields power no stronger than the Weisfeiler–Lehman (WL) test. In this work, we challenge this primitive. We introduce the Grothendieck Graph Neural Networks (GkGNN) framework, which provides a strict algebraic extension of neighborhoods to *covers*, and in doing so replaces neighborhoods as the fundamental objects of message passing. Neighborhoods and adjacency matrices are recovered as special cases, while covers enable a principled and flexible foundation for defining topology-aware propagation schemes. GkGNN formalizes covers and systematically translates them into matrices, analogously to how adjacency matrices encode neighborhoods, enabling both theoretical analysis and practical implementation. Within this framework, we introduce the *cover of sieves*, inspired by category theory, which captures rich topological structure. Based on this cover, we design *Sieve Neural Networks* (SNN), a canonical fixed-cover instantiation that generalizes the adjacency matrix. Experiments show that SNN achieves zero observed failures on challenging graph isomorphism benchmarks (SRG, CSL, BREC) and substantially improves topology-aware evaluation via a controlled label-propagation probe. These results establish GkGNN as a principled *foundational framework* for replacing neighborhoods in GNNs.

1 Introduction

The concept of *neighborhood* plays a central role in most Graph Neural Network (GNN) architectures, serving as the foundation of message passing (Gilmer et al., 2017). This reliance is not arbitrary: neighborhoods provide comprehensive coverage of the graph structure, leveraging the adjacency matrix to facilitate efficient and systematic aggregation of local information. However, this neighborhood-based perspective comes with intrinsic limitations. In particular, many GNNs have expressive power bounded by the WL test (Sato, 2020), (Xu et al., 2019), restricting their ability to capture broader topological structures.

To address the limitations of neighborhoods, researchers have proposed alternatives that incorporate richer structural information. One direction uses concepts from algebraic topology, such as simplicial complexes and higher-order faces, to capture interactions beyond pairs of nodes (Bodnar et al., 2021b), (Bodnar et al., 2021a), (Hajij et al., 2023), (Papillon et al., 2025). Another line of work relies on specific patterns or subgraphs (e.g., motifs) to encode characteristic structures as the basis for topologically-aware message passing (Bouritsas et al., 2023), (Ai et al., 2022). While these methods enrich the local perspective, they often depend on handcrafted definitions or combinatorial choices. In contrast, neighborhoods themselves arise from a precise algebraic definition, suggesting that a systematic algebraic formalism for extending neighborhoods may provide a broader and more principled foundation for message passing itself.

Viewpoint. We argue that neighborhoods should no longer be regarded as the primitive objects of message passing. Instead, we propose to replace neighborhoods by *covers*[1] as the fundamental units over which

[1] The term *cover* here should not be confused with graph-theoretic vertex or edge covers; it refers to a collection of algebraically defined message-passing strategies.

message passing operates. Covers strictly generalize neighborhoods: neighborhoods are recovered as a special case, while covers enable algebraically defined alternatives to neighborhood aggregation that go beyond locality. The purpose of this paper is to establish the algebraic and invariance foundations of this replacement. In particular, we introduce the first algebraic formalism for systematically generating and implementing infinitely many message-passing strategies, ranging from standard neighborhood aggregation to covers that encode richer topological structure. Covers may be instantiated as fixed algebraic objects, as in this work, or parameterized and learned in future architectures. Throughout this paper, replacing neighborhoods refers to replacing them as the primitive abstraction of message passing, not to discarding adjacency-based information: neighborhoods appear as specific elements within a larger algebraic space of composable propagation operators.

Building on this viewpoint, our work introduces an algebraic framework that extends neighborhoods to covers while preserving their structural role. This leads to the *Grothendieck Graph Neural Networks* (GkGNN) framework, an algebraic platform that formalizes covers, studies their invariance properties, and systematically translates them into matrix representations. In this framework, the adjacency matrix appears as a special case, while more expressive covers give rise to new message-passing operators.

Because this work extends the notion of neighborhoods, any existing GNN architecture can, in principle, be reformulated to operate on covers instead of neighborhoods. Accordingly, our focus is not on proposing a new learning algorithm, but on establishing the algebraic and invariance foundations that make such a replacement possible. At the core of GkGNN is a duality between graphs and algebraic structures, showing that graphs can be characterized by monoids generated from their covers.

Our main contributions are as follows.

- **Replacing neighborhoods by covers.** We introduce the concept of *covers* for graphs as a strict algebraic extension of neighborhoods. This generalization provides a principled and flexible foundation for representing graph structure.
- **The GkGNN framework.** We develop the *Grothendieck Graph Neural Networks* (GkGNN) framework, which creates, refines, and transforms covers into their matrix forms, recovering the adjacency matrix as a special case. GkGNN offers a systematic way to design new message-passing strategies.
- **Sieve Neural Networks (SNN).** As a concrete instantiation of GkGNN, we introduce *Sieve Neural Networks* (SNN) as a canonical example showing how expressive algebraic covers can be transformed into message-passing operators.
- **Topology-aware evaluation.** In addition to evaluating SNN on graph isomorphism tasks, we design a topology-encoding benchmark based on a special case of Label Propagation (LP) (Zhu & Ghahramani, 2002; Huang et al., 2020) with one propagation step ($\alpha = 1$). Here LP is used not as a learning algorithm but as a controlled probe to directly compare covers with neighborhoods. On citation networks and ogbn-arxiv, sieve covers consistently and substantially outperform the neighborhood cover, highlighting the advantage of covers in capturing topological structure in large graphs.

The proposed extension of the neighborhood concept in this paper relies on introducing four monoids (see Appendix A for formal definition of a monoid and monoidal homomorphism), $(\mathsf{Mod}(G), \bullet) \subseteq (\mathsf{SMult}(G), \bullet)$ and $(\mathsf{Mom}(G), \circ) \subseteq (\mathsf{Mat}_n(\mathbb{R}), \circ)$. In this setting, neighborhoods are elements of $\mathsf{Mod}(G)$, and a monoidal homomorphism $\mathsf{Tr} : \mathsf{Mod}(G) \to \mathsf{Mom}(G)$ translates elements of $\mathsf{Mod}(G)$ into matrices.

This organization allows us to treat graphs as algebraic spaces, in a manner analogous to familiar examples such as $(\mathbb{N}, \times)$, where natural numbers are composed by multiplication, or $(\mathsf{Mat}_n(\mathbb{R}), \cdot)$, where matrices are composed by matrix multiplication. In the same spirit, the monoid operation $\bullet$ enables algebraic compositions of graph-derived objects, for instance $N(u) \bullet N(v)$.

As proved in Theorem 3.2.5, directed edges generate $\mathsf{Mod}(G)$. Intuitively, directed edges act as elementary building blocks: by composing edges as $e_1 \bullet e_2 \bullet \cdots \bullet e_k$, one can generate both neighborhoods and elements

beyond neighborhoods, in an informal sense analogous to how the monoid $(\mathbb{N}, \times)$ is generated by prime numbers under multiplication.

The map Tr plays an essential role in the GkGNN framework: it is not merely a function, but a monoidal homomorphism. For example, the determinant defines a monoidal homomorphism from $(\mathsf{Mat}_n(\mathbb{R}), \cdot)$ to $(\mathbb{R}, \times)$, since $\det(A \cdot B) = \det(A) \times \det(B)$. Since Tr is a monoidal homomorphism, the translation of a composed element $e_1 \bullet e_2 \bullet \cdots \bullet e_k \in \mathsf{Mod}(G)$ can be computed compositionally as $\mathsf{Tr}(e_1 \bullet e_2 \bullet \cdots \bullet e_k) = \mathsf{Tr}(e_1) \circ \mathsf{Tr}(e_2) \circ \cdots \circ \mathsf{Tr}(e_k)$, which preserves the algebraic structure of compositions when passing from graph-based objects to matrices.

2 Related work

Many classical GNN architectures can be unified under the neighborhood-based *Message Passing Neural Network* (MPNN) paradigm Gilmer et al. (2017). A large body of work seeks to move beyond strict 1-hop neighborhoods by altering the graph on which messages are passed or by enriching the operators/features used for aggregation.

Passing messages on derived graphs. One line of work replaces the original graph with a derived graph and then applies MPNN. For example, Gasteiger et al. (2021) constructs the *directed line graph*, whose nodes correspond to directed edges of the original graph and where two nodes are adjacent if the underlying edges share an endpoint; message passing is then performed on this derived graph. In Ai et al. (2022), each graph is mapped to a *topology-level* summary graph built from subgraphs; message passing runs jointly on the original graph and its summary, making the propagation explicitly topology-aware.

Substructure and kernel based encodings. Another direction injects information about motifs or subgraphs. Graph Substructure Networks (GSNs) Bouritsas et al. (2023) enrich node/edge features with positions within selected patterns, integrating substructure signals into message passing. KerGNNs Feng et al. (2022a) use small graphs as filters—via graph kernels such as random-walk kernels—applied to node-centered subgraphs; replacing the raw neighborhood with a filtered subgraph can increase expressivity over vanilla MPNNs.

Contextual and multi-hop neighborhoods. Contextualization beyond the immediate neighborhood is also common. ID-GNN You et al. (2021) attends to occurrences of a node within its ego network, effectively differentiating its roles across contexts. Extensions to k-hop neighborhoods Feng et al. (2022b) aggregate information from larger receptive fields; KP-GNN further selects k-hop neighbors via shortest-path or random-walk kernels, yielding a framework that can surpass standard MPNNs.

Local topology operators and stochastic perturbations. In Vignac et al. (2020), each node is associated with a *local context* matrix intended to capture surrounding topology; these contexts replace raw features during message passing and have been shown effective on topology-sensitive tasks (e.g., cycle detection), outperforming MPNNs in those settings. The approach in Papp et al. (2021) applies message passing to randomly thinned graphs obtained by deleting each node with small probability and aggregates the outcomes, preserving much of the original topology while introducing beneficial stochasticity.

Topological deep learning. Tools from algebraic topology provide higher-order generalizations of graphs that encode multi-level interactions. Works based on simplicial and CW complexes Bodnar et al. (2021b;a) replace node–edge neighborhoods with higher-dimensional cells and associated incidence structures, yielding message-passing schemes that explicitly reason over topology beyond pairwise relations.

3 Covers and their matrix interpretations

In this section we develop the notion of *covers* for graphs and show how to interpret them as matrices, laying the groundwork for the GkGNN framework. We begin by assigning to each directed subgraph its matrix representation, establishing a bijection between directed subgraphs and their associated matrices. We then

introduce two monoids: $\mathsf{Mod}(G)$, generated by directed subgraphs, and $\mathsf{Mom}(G)$, generated by their matrix representations. This allows us to extend the representation map to a monoidal homomorphism

$$\mathsf{Tr}: \ \mathsf{Mod}(G) \ \longrightarrow \ \mathsf{Mom}(G).$$

We prove that Tr is invariant under graph isomorphisms (via Change-of-Order mappings) and provides an algebraic description of a graph that is unique up to isomorphism. These results form the theoretical foundation of the GkGNN framework.

3.1 Matrix representations of directed subgraphs

We consider undirected graphs $G = (V, E)$ whose node set V is equipped with a fixed ordering. Our first step is to formalize *directed subgraphs* of G and to define their matrix representations.

Definition 3.1.1. *(1) A* ***path*** *p from node v_{p_1} to node v_{p_m} is an ordered sequence*

$$v_{p_1}, e_{p_1}, v_{p_2}, e_{p_2}, \ldots, v_{p_{m-1}}, e_{p_{m-1}}, v_{p_m},$$

where each e_{p_i} connects v_{p_i} and $v_{p_{i+1}}$.

(2) A ***directed subgraph*** *D of G is a connected, acyclic subgraph in which each edge is assigned a* direction.

Neighborhoods as a special case. A neighborhood can be seen as a directed subgraph obtained by orienting all incident edges *into* a fixed node (see Figure 5). In the adjacency matrix, each column corresponds to such a neighborhood; isolating the neighborhood of a node amounts to zeroing out the other columns.

Matrix representation of a directed subgraph. We extend the neighborhood-as-column view to arbitrary directed subgraphs by encoding *direction-respecting reachability.*

Definition 3.1.2. *Let D be a directed subgraph of $G = (V, E)$.*

1. *Define a relation $\leq_D$ on V by $v_i \leq_D v_j$ iff there exists a path in D that* respects edge directions *and starts at v_i and ends at v_j.*
2. *The* matrix representation *of D is the $|V| \times |V|$ matrix M_D with $(M_D)_{ij} = 1$ if $v_i \leq_D v_j$ and $(M_D)_{ij} = 0$ otherwise.*

The requirement that paths respect directions is essential: all walks counted by M_D must follow the edge orientations in D. Intuitively, a directed subgraph specifies an *allowable message-flow pattern*; its matrix M_D realizes this pattern. In this sense, matrices from directed subgraphs serve as structured alternatives to the standard adjacency matrix used in neighborhood-based message passing.

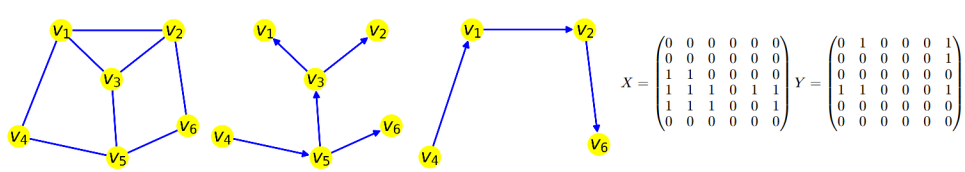

Figure 1: **Directed subgraphs and their matrices.** Two directed subgraphs of a graph G (left) are shown: $\hat{D}$ (middle) and $\bar{D}$ (right). Their matrix representations are X and Y, respectively. Each directed subgraph encodes a distinct strategy for propagating information; the matrices X and Y make these strategies directly usable in message passing.

Representation map. Definition 3.1.2 induces a map from directed subgraphs to matrices:

$$\mathsf{Rep}: \ \mathsf{DirSub}(G) \ \longrightarrow \ \mathsf{MatRep}(G),$$

where $\mathsf{DirSub}(G)$ is the set of directed subgraphs of G and $\mathsf{MatRep}(G)$ is the image (subset) of $\mathsf{Mat}_{|V|}(\mathbb{R})$ consisting of matrices that arise from directed subgraphs via Definition 3.1.2.

Theorem 3.1.3. *The map* Rep *is an isomorphism between* $\mathsf{DirSub}(G)$ *and* $\mathsf{MatRep}(G)$*. In particular, each directed subgraph is uniquely determined by its matrix representation, and conversely every matrix in* $\mathsf{MatRep}(G)$ *corresponds to a unique directed subgraph.*

3.2 Defining covers for graphs: an algebraic platform

While we can cover a graph G by picking elements from $\mathsf{DirSub}(G)$ and map each to a matrix via Rep, this space is limited: $\mathsf{DirSub}(G)$ is relatively small and its elements do not combine well. For instance, the union of the directed subgraphs $\hat{D}$ and $\bar{D}$ in Figure 1 is *not* a directed subgraph (multiple directed paths appear between some node pairs). Hence no matrix image exists for such a combination, which prevents its direct use in a message-passing scheme. This makes it hard to design diverse, meaningful strategies using only $\mathsf{DirSub}(G)$.

Step 1: enlarge the space via a multigraph monoid. To combine directed subgraphs systematically, we first endow them with an algebraic operation. A natural choice is to define $C \bigoplus D$ for $C, D \in \mathsf{DirSub}(G)$ as the *directed multigraph* obtained by taking the union of their node sets and the disjoint union of their directed edge sets. Let

$$\mathsf{Mult}(G) = \Big\{ \bigoplus_{i=1}^{k} D_i \ : \ k \geq 1, \ D_i \in \mathsf{DirSub}(G) \Big\}.$$

Then $(\mathsf{Mult}(G), \bigoplus)$ is a commutative monoid. This construction enlarges the object set, but it treats edges as independent and is largely *insensitive to path structure*: paths in $C \bigoplus D$ are simply concatenations of available directed edges, with little inheritance from the path sets of C and D. As a result, $\bigoplus$ alone provides limited control for crafting new message-passing strategies.

Step 2: add path sensitivity via a non-commutative monoid. To encode which *paths* are allowed, not only which edges exist, we augment multigraphs with explicit path sets. Define

$$\mathsf{SMult}(G) \ = \ \big\{ (M, S) \ : \ M \in \mathsf{Mult}(G), \ S \subseteq \mathsf{Paths}(M) \big\},$$

and define a binary operation $\bullet$ by $(M, S) \bullet (N, T) \ = \ \big(M \bigoplus N, \ S \star T\big)$, where $S \star T$ is the union of the sets S, T, and the collection of paths constructed by the composition of paths in S followed by paths in T. Since edges of $M \bigoplus N$ come from a disjoint union, paths in S and T remain disjoint subsets of $\mathsf{Paths}(M \bigoplus N)$.

Theorem 3.2.1. $(\mathsf{SMult}(G), \bullet)$ *is a non-commutative monoid.*

Non-commutativity comes from the order of path composition inside $\star$: if (M, S) and (N, T) have composable paths, then generally $(M, S) \bullet (N, T) \neq (N, T) \bullet (M, S)$; if they have no composable paths, $\star$ reduces to set union.

Example 3.2.2. *Let* $d : u \to v$ *and* $e : v \to w$ *be directed edges. Then* $d \bullet e$ *and* $e \bullet d$ *share the same edge multiset* $(d \bigoplus e)$*, but differ in allowed paths:* $d \bullet e$ *contains a path from* u *to* w *(via* d *then* e*), whereas* $e \bullet d$ *does not. Thus the order of composition matters.*

Step 3: a representable submonoid for covers. Elements $(M, S) \in \mathsf{SMult}(G)$ can be read as *message-passing strategies*: M fixes the directed multigraph over which messages may travel, and S specifies which directed paths are allowed. However, not every such pair admits a matrix representation compatible with an extension of Rep. To retain implementability, we restrict to the submonoid generated by genuine directed subgraphs, embedded via $D \mapsto \big(D, \mathsf{Paths}(D)\big)$.

Definition 3.2.3. *For a graph* G*, the* monoid of directed subgraphs *is the submonoid* $\mathsf{Mod}(G) \subseteq \mathsf{SMult}(G)$ *generated by* $\mathsf{DirSub}(G)$*.*

Hence each $(M, S) \in \mathsf{Mod}(G)$ can be written as

$$(M, S) \ = \ D_1 \bullet \cdots \bullet D_k, \qquad M \ = \ \bigoplus_{i=1}^{k} D_i, \qquad S \ = \ \mathsf{Paths}(D_1) \star \cdots \star \mathsf{Paths}(D_k).$$

In the next subsection we show that all elements of $\mathsf{Mod}(G)$ admit matrix representations via an extension of Rep, which makes them suitable for implementation.

Definition 3.2.4. *A* cover *of G is a finite collection of elements of* $\mathsf{Mod}(G)$.

A cover thus specifies a family of message-passing strategies: each element constrains allowed paths locally, and the collection provides a flexible, task-dependent view of the graph (we do not require covering all nodes/edges). Because $\mathsf{Mod}(G)$ is infinite and $\bullet$ is non-commutative, this space is expressive enough to encode a wide range of perspectives on how information should flow. Moreover, the following result shows that directed edges suffice as generators, so complex strategies can be built compositionally.

Theorem 3.2.5. *Directed edges generate* $\mathsf{Mod}(G)$.

3.3 From covers to matrices: an algebraic perspective

Our goal in this section is to extend the representation map Rep from directed subgraphs to arbitrary elements of $\mathsf{Mod}(G)$, so that a *cover* can be transformed into a *collection of matrices*. This requires a matrix-side operation that mirrors the path-composition on $\mathsf{Mod}(G)$.

A monoid on matrices. Let $\mathsf{Mat}_n(\mathbb{R})$ be the set of $n \times n$ real matrices. Define a binary operation

$$A \circ B \;:=\; A \;+\; B \;+\; AB.$$

Theorem 3.3.1. $(\mathsf{Mat}_n(\mathbb{R}), \circ)$ *is a monoid.*

We now take the submonoid *generated* by $\mathsf{MatRep}(G)$ inside $(\mathsf{Mat}_{|V|}(\mathbb{R}), \circ)$.

Definition 3.3.2. *The* monoid of matrix representations *of G is the submonoid* $(\mathsf{Mom}(G), \circ)$ *of* $(\mathsf{Mat}_{|V|}(\mathbb{R}), \circ)$ *generated by* $\mathsf{MatRep}(G)$.

Extending Rep to covers. Elements of $\mathsf{Mod}(G)$ are built by composing directed subgraphs with $\bullet$. The map below sends such compositions to matrices by replacing each directed subgraph D_i with its matrix $\mathsf{Rep}(D_i)$ and each $\bullet$ with $\circ$.

Theorem 3.3.3. *The mapping* $\mathsf{Tr} : \mathsf{Mod}(G) \longrightarrow \mathsf{Mom}(G)$,

$$(M, S) = D_1 \bullet D_2 \bullet \cdots \bullet D_k \longmapsto A \;=\; A_1 \circ A_2 \circ \cdots \circ A_k,$$

where $D_i \in \mathsf{DirSub}(G)$ and $A_i = \mathsf{Rep}(D_i)$, is a surjective monoidal homomorphism.

Interpretation (path counting). In the proof of Theorem 3.3.3 one sees that Tr acts as a *path counter*: for $(M, S) \in \mathsf{Mod}(G)$, the (i, j) entry of $\mathsf{Tr}(M, S)$ equals the number of paths in S from v_i to v_j. Thus, covers become *collections of matrices* in the same way that neighborhoods give rise to the adjacency matrix, but now with explicit control over composed paths. Although we were not able to show that Tr is an isomorphism, it extends an isomorphism on $\mathsf{DirSub}(G)$ and will be shown in the next subsection to characterize graphs up to isomorphism, supporting its use as a faithful matrix transformation of covers.

Arising from Theorem 3.3.3, the following commutative diagram illustrates how Tr extends the representation map Rep:

$$\begin{tikzcd}
\mathsf{DirSub}(G) \arrow[r, "\mathsf{Rep}"] \arrow[d, hook] & \mathsf{MatRep}(G) \arrow[d, hook] \\
\mathsf{Mod}(G) \arrow[r, "\mathsf{Tr}"'] & \mathsf{Mom}(G)
\end{tikzcd}$$

Since $\mathsf{Rep} : \mathsf{DirSub}(G) \to \mathsf{MatRep}(G)$ is an isomorphism, Tr is constructed to agree with Rep on directed subgraphs: under the canonical embedding $D \mapsto (D, \mathrm{Paths}(D)) \in \mathsf{Mod}(G)$, we have $\mathsf{Tr}(D) = \mathsf{Rep}(D)$. Thus Tr extends an isomorphism on $\mathsf{DirSub}(G)$ while enlarging the space to $\mathsf{Mod}(G)$.

The non-commutativity of $\bullet$ (and correspondingly of $\circ$) is essential for this extension to remain meaningful beyond $\mathsf{DirSub}(G)$, because ordered compositions encode order of directed edges in paths. To see this, let $D \in \mathsf{DirSub}(G)$ be the path

$$u \xrightarrow{d} v \xrightarrow{e} w \xrightarrow{f} z\ .$$

In $\mathsf{Mod}(G)$, the same object can be written using edge generators as $D = d \bullet e \bullet f = (M, S)$, where

$$S = \{d, e, f, de, ef, def\},$$

and de, ef, def denote successive compositions of edges (subpaths of D). By monoidality,

$$\mathsf{Tr}(D) = \mathsf{Tr}(d \bullet e \bullet f) = \mathsf{Tr}(d) \circ \mathsf{Tr}(e) \circ \mathsf{Tr}(f),$$

and Tr records exactly the paths in $S = \mathrm{Paths}(D)$ (via its path-counting interpretation), hence $\mathsf{Tr}(D) = \mathsf{Rep}(D)$ on $\mathsf{DirSub}(G)$. If $\bullet$ were commutative, ordered compositions such as $d \bullet e$ and $e \bullet d$ would coincide, collapsing distinct path constraints and destroying the order-sensitive information that Tr is meant to carry when extending Rep beyond single directed subgraphs.

Example 3.3.4. *In Figure 1, let $\hat{D}$ and $\bar{D}$ be two directed subgraphs of G with matrix representations X and Y. Using $\bullet$, we may form new strategies $\hat{D} \bullet \bar{D}$ and $\bar{D} \bullet \hat{D}$. Their matrix transforms are obtained as follows:*

$$\mathsf{Tr}(\hat{D} \bullet \bar{D}) \;=\; \mathsf{Tr}(\hat{D}) \circ \mathsf{Tr}(\bar{D}) \;=\; X \circ Y, \qquad \mathsf{Tr}(\bar{D} \bullet \hat{D}) \;=\; \mathsf{Tr}(\bar{D}) \circ \mathsf{Tr}(\hat{D}) \;=\; Y \circ X.$$

Since $\circ$ is generally non-commutative in this context, the two results differ, reflecting the order-sensitivity of path composition in the underlying cover construction.

4 Grothendieck Graph Neural Networks Framework

4.1 Algebraic foundations of graphs

We have introduced two monoids associated with a graph and a monoidal homomorphism between them. We now ask: *to what extent do these algebraic objects describe the underlying graph?* To address this, we first formalize how reordering node indices acts on the matrix space and verify that this action is compatible with our monoidal structures.

Change-of-Order mappings. A matrix $A \in \mathsf{Mat}_n(\mathbb{R})$ represents a linear map $\mathbb{R}^n \to \mathbb{R}^n$ with respect to the standard basis. If we reorder the standard basis of $\mathbb{R}^n$ (equivalently, relabel the coordinates), the matrix representation of the same linear map is obtained by reindexing the rows and columns of A. Any linear isomorphism $f : \mathsf{Mat}_n(\mathbb{R}) \to \mathsf{Mat}_n(\mathbb{R})$ arising in this way is called a **Change-of-Order mapping** (see Example A.2.1). Intuitively, this captures the effect of node relabeling at the matrix level.

Proposition 4.1.1. *Suppose $f : \mathsf{Mat}_n(\mathbb{R}) \to \mathsf{Mat}_n(\mathbb{R})$ is a Change-of-Order mapping. Then f preserves the standard algebraic operations on matrices: it is compatible with monoidal operation $\circ$, with matrix multiplication, and with element-wise (Hadamard) multiplication.*

Relating graph isomorphisms and algebra. A graph isomorphism $f : G \to H$ is a bijection on nodes that preserves edges, and therefore corresponds to a reordering of node indices. Hence it induces a Change-of-Order mapping $\mathsf{CO}(f) : \mathsf{Mat}_{|V_G|}(\mathbb{R}) \to \mathsf{Mat}_{|V_H|}(\mathbb{R})$. The next result shows that this relabeling is compatible with our monoidal constructions and with the translation to matrices.

Theorem 4.1.2. *Every graph isomorphism $f : G \to H$ induces monoidal isomorphisms $\mathsf{Mod}(f) : \mathsf{Mod}(G) \to \mathsf{Mod}(H)$ and $\mathsf{Mom}(f) : \mathsf{Mom}(G) \to \mathsf{Mom}(H)$ such that the following diagram commutes, where ι denotes the inclusions:*

$$\begin{tikzcd}
\mathsf{Mod}(G) \arrow[r, "\mathsf{Tr}_G"] \arrow[d, "\mathsf{Mod}(f)"'] & \mathsf{Mom}(G) \arrow[r, hook, "\iota"] \arrow[d, "\mathsf{Mom}(f)"'] & \mathsf{Mat}_{|V_G|}(\mathbb{R}) \arrow[d, "\mathsf{CO}(f)"] \\
\mathsf{Mod}(H) \arrow[r, "\mathsf{Tr}_H"'] & \mathsf{Mom}(H) \arrow[r, hook, "\iota"'] & \mathsf{Mat}_{|V_H|}(\mathbb{R})
\end{tikzcd} \tag{1}$$

A converse direction. We also have a partial converse: if a Change-of-Order mapping identifies the matrix-level monoids of two graphs, then the graphs are isomorphic.

Theorem 4.1.3. *Let G and H be graphs with $|V_G| = |V_H| = n$, and let $f : \mathsf{Mat}_n(\mathbb{R}) \to \mathsf{Mat}_n(\mathbb{R})$ be a Change-of-Order mapping. If the restriction of f to $\mathsf{Mom}(G)$ is an isomorphism onto $\mathsf{Mom}(H)$, then G and H are isomorphic.*

These theorems establish a duality between graphs and monoids, in which elements of $\mathsf{Mod}(G)$ encode structural relations intrinsic to the graph. Importantly, this duality does not originate from neighborhoods alone, since neighborhoods are known to be insufficient to characterize graphs, but from richer elements of $\mathsf{Mod}(G)$ that capture more global, structure-aware information.

4.2 Definition of the GkGNN framework

Theorems 4.1.2 and 4.1.3 establish the key invariance principle underlying our construction. In Diagram 1, an isomorphism between graphs G and H corresponds to the vertical homomorphisms being isomorphisms; equivalently, a relabeling of nodes in a graph induces isomorphic transformations both on a cover and on its matrix transformation. Consequently, the *horizontal* arrows in the diagram provide an algebraic description that is unique up to graph isomorphism. Leveraging this observation, we formalize the *Grothendieck Graph Neural Networks (GkGNN)* framework as the following algebraic pipeline:

Definition 4.2.1. *For a graph $G = (V, E)$, the GkGNN framework is the composition*

$$\mathsf{Mod}(G) \xrightarrow{\mathsf{Tr}} \mathsf{Mom}(G) \xhookrightarrow{\iota} \mathsf{Mat}_{|V|}(\mathbb{R}) \tag{2}$$

How GkGNN is used. The framework exposes three actions: (i) *choose* a cover in $\mathsf{Mod}(G)$; (ii) *translate* it to matrices via Tr; (iii) optionally *enrich* the resulting collection inside $\mathsf{Mat}_{|V|}(\mathbb{R})$ using the allowed operations from Proposition 4.1.1. See Appendix B for more details.

Neighborhoods as a special case. The framework recovers standard message passing from neighborhoods:

Theorem 4.2.2. *The collection of neighborhoods forms a cover in $\mathsf{Mod}(G)$ and, under Tr, maps to the adjacency matrix in $\mathsf{Mat}_{|V|}(\mathbb{R})$.*

Remarks. (i) A graph can be characterized by the submonoid generated by its neighborhoods; see Appendix E.1. (ii) We compare GkGNN with higher-order GNN families in Appendix C.

5 Sieve Neural Networks: a model within the GkGNN framework

We emphasize that Sieve Neural Networks (SNN) is not the goal of GkGNN, but a canonical example demonstrating how expressive algebraic elements of $\mathsf{Mod}(G)$ can be identified and transformed into concrete message-passing operators. SNN is deliberately fixed and non-learned, in order to isolate the representational effect of covers and to exemplify how elements beyond neighborhoods can be used in practice.

Inspired by *sieves* in category theory (MacLane & Moerdijk, 1994), we instantiate the GkGNN framework with a concrete cover that yields the *Sieve Neural Network*. For each node v, we construct a family of outward-expanding substructures, formalized as elements of $\mathsf{Mod}(G)$, which together form a *sieve-inspired cover* of the graph. This cover generalizes the standard neighborhood view by admitting multiple direction-respecting pathways for information exchange. Via the matrix map Tr, the cover is translated into operators used for message passing, exposing richer topological relationships than adjacency-based aggregation while remaining permutation-consistent. In what follows, we define the sieve and cosieve elements, assemble the cover, and derive the SNN architecture from their matrix transformations.

5.1 Cover of sieves for graphs

Constructing sieve elements in $\mathsf{Mod}(G)$. Fix a node v and build breadth-first "layers" around v:

$$N_0(v) = \{v\}, \qquad N_1(v) = N(v), \qquad N_k(v) = \Big(\bigcup_{u \in N_{k-1}(v)} N(u)\Big) \setminus \bigcup_{i=0}^{k-1} N_i(v) \quad (k \geq 2).$$

For each $k \geq 1$, orient all edges *toward* v across consecutive layers and collect them as

$$M_k(v) = \{\, w \to u \;:\; wu \in E,\; w \in N_k(v),\; u \in N_{k-1}(v) \,\}, \qquad M_0(v) = \varnothing.$$

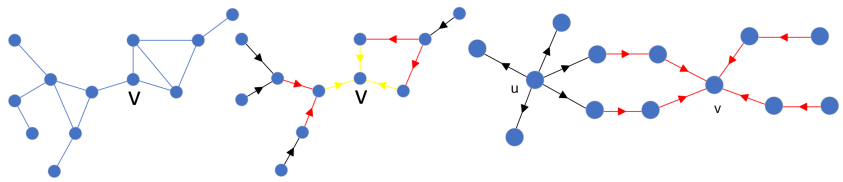

Figure 2: **Sieve construction.** Left: a graph G. Middle: $\mathsf{Sieve}(v,3) = D_3(v) \bullet D_2(v) \bullet D_1(v)$ around v; yellow, red, and black edges indicate $D_1(v)$, $D_2(v)$, and $D_3(v)$, respectively. Right: a graph H; the element $\mathsf{CoSieve}(u,1) \bullet \mathsf{Sieve}(v,2) \in \mathsf{Mod}(H)$ specifies allowed interactions between u and v in $\mathsf{SNN}(\alpha,(1,2))$.

Edges in the same $M_k(v)$ are pairwise non-composable (each goes from layer k to $k-1$), so the order in which they are combined is irrelevant. Define

$$D_k(v) \;:=\; \bullet_{e \in M_k(v)} e$$

and assemble the depth-k *sieve element*

$$\mathsf{Sieve}(v,k) \;:=\; D_k(v) \bullet D_{k-1}(v) \bullet \cdots \bullet D_1(v) \bullet D_0(v),$$

where $D_0(v)$ is the identity of $\mathsf{Mod}(G)$; see Figure 2. Since the layers eventually empty, there exists k_0 with $N_{k_0+1}(v) = \varnothing$, hence $\mathsf{Sieve}(v,k)$ stabilizes for $k \geq k_0$. We denote this saturated element by $\mathsf{Sieve}(v,-1) := \mathsf{Sieve}(v,k_0)$. To construct the *opposite* sieve, reverse the directions in each $M_k(v)$: let $M_k^{\mathrm{op}}(v)$ be $M_k(v)$ with all edges reversed and set

$$D_k^{\mathrm{op}}(v) \;:=\; \bullet_{e \in M_k^{\mathrm{op}}(v)} e, \qquad \mathsf{CoSieve}(v,\ell) \;:=\; D_0^{\mathrm{op}}(v) \bullet D_1^{\mathrm{op}}(v) \bullet \cdots \bullet D_\ell^{\mathrm{op}}(v).$$

The cover of sieves. For every node v in G, collect all depth-truncated and saturated sieve and co-sieve elements:

$$\mathsf{Sieve}(v,0),\ \mathsf{Sieve}(v,1),\ \ldots,\ \mathsf{Sieve}(v,-1) \quad \text{and} \quad \mathsf{CoSieve}(v,0),\ \mathsf{CoSieve}(v,1),\ \ldots,\ \mathsf{CoSieve}(v,-1).$$

The *cover of sieves* is the finite collection containing these elements for all nodes v in G.

Matrix interpretation of the cover of sieves. Apply the monoidal homomorphism Tr to obtain matrices:

$$\mathsf{Image}(v,k) := \mathsf{Tr}\big(\mathsf{Sieve}(v,k)\big), \qquad \mathsf{CoImage}(v,\ell) := \mathsf{Tr}\big(\mathsf{CoSieve}(v,\ell)\big).$$

Since Tr is monoidal,

$$\begin{aligned} \mathsf{Image}(v,k) &= \mathsf{Tr}\big(D_k(v) \bullet D_{k-1}(v) \bullet \cdots \bullet D_0(v)\big) \\ &= \mathsf{Tr}\big(D_k(v)\big) \circ \mathsf{Tr}\big(D_{k-1}(v)\big) \circ \cdots \circ \mathsf{Tr}\big(D_0(v)\big). \end{aligned} \tag{3}$$

Within a fixed layer i, edges in $M_i(v)$ are not composable, so for distinct $e, c \in M_i(v)$ we have $\mathsf{Tr}(e)\,\mathsf{Tr}(c) = \mathsf{Tr}(c)\,\mathsf{Tr}(e) = 0$. Hence, by Theorem F.5.1,

$$\mathsf{Tr}\big(D_i(v)\big) = \mathsf{Tr}\Big(\bullet_{e \in M_i(v)} e\Big) = \circ_{e \in M_i(v)} \mathsf{Tr}(e) = \sum_{e \in M_i(v)} \mathsf{Tr}(e),$$

i.e., $\mathsf{Tr}\big(D_i(v)\big)$ is obtained from the adjacency matrix by keeping only entries corresponding to directed edges in $M_i(v)$. Moreover, $\mathsf{CoImage}(v,\ell) = \mathsf{Image}(v,\ell)^\top$ (transpose), so it suffices to compute one of them. Algorithms for these computations are given in Appendix D.4. In addition, Appendix E.2 shows that the submonoid generated by this cover determines the graph.

Invariance. The cover of sieves is stable under graph isomorphisms, and the induced matrices transform accordingly.

Theorem 5.1.1. *If $f : G \to H$ is a graph isomorphism, then* $\mathsf{Mod}(f)\big(\mathsf{Sieve}(v,k)\big) = \mathsf{Sieve}\big(f(v),k\big)$ *and* $\mathsf{Mom}(f)\big(\mathsf{Image}(v,k)\big) = \mathsf{Image}\big(f(v),k\big)$.

5.2 Design and construction of the model

Building on the cover of sieves and its matrix interpretation, we define the *Sieve Neural Network* (SNN) in two variants of increasing flexibility. A detailed comparison with MPNNs is provided in Appendix D.

Variant $\mathsf{SNN}(\alpha, (l, k))$**.** In the α variant, for each ordered pair of nodes (v_i, v_j) we interpret $\mathsf{CoSieve}(v_i, l)$ as a *sender* and $\mathsf{Sieve}(v_j, k)$ as a *receiver*. The admissible transmissions from v_i to v_j are precisely the paths allowed by the composed element $\mathsf{CoSieve}(v_i, l) \bullet \mathsf{Sieve}(v_j, k)$, (see Figure 2). Under the map Tr, the number of such paths equals the (i, j) entry of $\mathsf{CoImage}(v_i, l) \circ \mathsf{Image}(v_j, k)$. To obtain a scale-aware score, we normalize by the *sending capacity* of v_i and the *receiving capacity* of v_j: let

$$r_i = \sum_q \big(\mathsf{CoImage}(v_i, l)\big)_{iq}, \qquad c_j = \sum_p \big(\mathsf{Image}(v_j, k)\big)_{pj}.$$

Then the output matrix of $\mathsf{SNN}(\alpha, (l, k))$ on G is $S^{(\alpha)}$ with $S^{(\alpha)}_{ij} = \frac{\left(\mathsf{CoImage}(v_i,l) \circ \mathsf{Image}(v_j,k)\right)_{ij}}{r_i\, c_j}$. Intuitively, $S^{(\alpha)}_{ij}$ is the fraction of realized paths relative to a node-pair–specific capacity. If the normalization is omitted, we denote the model by $\mathsf{SNN}_o(\alpha, (l, k))$.

Variant $\mathsf{SNN}(\beta, (l_1, \ldots, l_t))$**.** The β variant aggregates information globally by summing over node-centered images and co-images, alternating sender/receiver roles across depths. The model output is $S^{(\beta)} = Su_1 \circ Su_2 \circ \cdots \circ Su_t$, where $Su_i = \sum_{v \in V} \mathsf{CoImage}(v, l_i)$ for odd i and $Su_i = \sum_{v \in V} \mathsf{Image}(v, l_i)$ for even i, with $1 \leq i \leq t$.

How SNN is used. SNN is applied *once* as a preprocessing step to transform each graph: we replace its adjacency matrix with the corresponding SNN output (either $S^{(\alpha)}$ or $S^{(\beta)}$). The transformed dataset can then be fed to any message-passing GNN, substituting the traditional neighborhood cover with the sieve cover. The *time complexity* of this transformation is analyzed in Appendix D.3.

Invariance. The model respects graph isomorphisms (node relabelings), making it suitable for graph isomorphism and classification tasks.

Theorem 5.2.1. SNN *is invariant under graph isomorphism.*

5.3 Expressivity and WL

The main theoretical contribution of this paper is the duality established between graphs and monoids. In Theorems 4.1.2 and 4.1.3 we show that the monoidal structures $\mathsf{Mod}(G)$ and $\mathsf{Mom}(G)$, together with the homomorphism Tr, characterize graphs up to isomorphism. Since GkGNN is an algebraic design framework rather than a bounded-dimensional color-refinement procedure, it is not formulated within the k-WL hierarchy. Instead, it introduces a different axis of expressivity grounded in monoidal compositions of directed subgraphs and their matrix realizations. For the sieve construction, we further prove that the submonoid generated by the cover of sieves also characterizes graphs up to isomorphism (Appendix E.2).

While the concrete model SNN realizes a restricted family of elements from this submonoid, we can establish a principled bridge between these algebraic path compositions and bounded-dimensional refinement by examining the initialization phase of the k-WL test.

In the standard k-WL test, the initial hashing of node k-tuples plays an essential role in determining the algorithm's expressive bounds. This classical hashing mechanism is fundamentally based on the **cover of neighborhoods**, as it evaluates immediate adjacencies to capture the atomic isomorphism type of a tuple. Specifically, it requires identifying whether an edge exists between v_i and v_j for any i, j. Within our framework, this is algebraically equivalent to evaluating if there exists any valid path between v_i and v_j in the composed element $\mathsf{CoSieve}(v_i, 1) \bullet \mathsf{Sieve}(v_j, 0)$. Here, we adapt the k-WL algorithm to leverage the **cover of sieves** rather than standard neighborhoods. Inspired by this direct equivalence, we define a natural generalization:

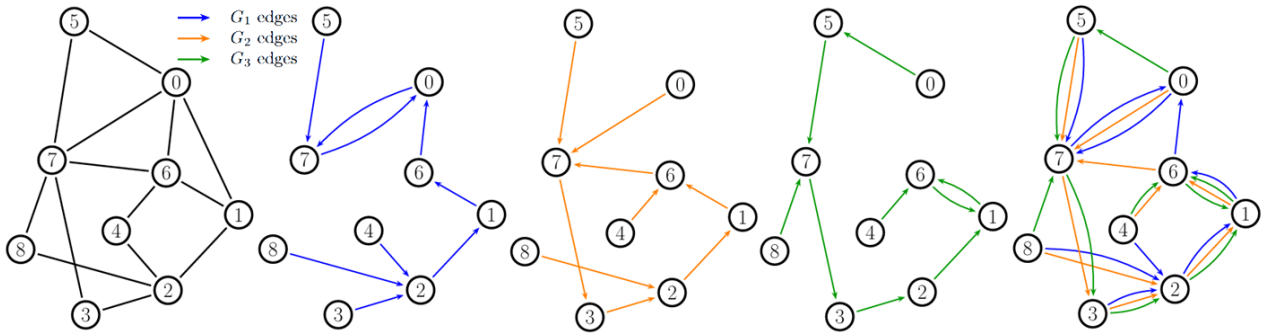

Figure 3: **Random walk–based monoidal construction.** For a graph G (left), we construct three directed multigraphs G_1 (blue), G_2 (orange), and G_3 (green) by applying Steps 1–2 of Example 5.4.1. Their multigraph sum $M = G_1 \oplus G_2 \oplus G_3$ is shown on the right. Let $\hat{G}_1, \hat{G}_2, \hat{G}_3$ be as in Example 5.4.1 (i.e., $\hat{G}_i = (G_i, E(G_i))$), and set $(M, S) = \hat{G}_1 \bullet \hat{G}_2 \bullet \hat{G}_3 \in \mathsf{SMult}(G)$. Then S consists of admissible directed paths of length at most 3 obtained by composing *at most one edge from each* G_i in the order $G_1 \prec G_2 \prec G_3$: length-2 paths are composable pairs with strictly increasing colors (blue→orange, blue→green, or orange→green), and length-3 paths are composable triples blue→orange→green. In particular, repeated colors (e.g., blue→blue) and any backward color order (e.g., orange→blue or green→orange) are not allowed. This figure illustrates how the monoidal product $\bullet$ enforces ordered path composition.

Definition 5.3.1. ***Sieve-adapted*** (r, s, k)***-WL algorithm.*** *The Sieve-adapted* (r, s, k)*-WL algorithm operates identically to the standard* k*-WL algorithm, with the exception that for each tuple* $\mathbf{v} = (v_1, \cdots, v_k)$*, its initial hash function is enriched to capture all paths between* v_i *and* v_j *residing within the algebraic element* $\mathsf{CoSieve}(v_i, r) \bullet \mathsf{Sieve}(v_j, s)$*, in addition to the tuple* $\mathbf{v}$ *itself.*

Under this definition, it is obvious that the $(1, 0, k)$-WL algorithm, which restricts its view to the cover of neighborhoods, is strictly equivalent to the classical k-WL algorithm. Furthermore, based on the definition of the monoidal operation $\bullet$ and the construction of the elements $\mathsf{Sieve}(v_i, r)$ and $\mathsf{CoSieve}(v_j, s)$, one can easily verify a property of monotonicity: for any $r \leq r'$ and $s \leq s'$, the element $\mathsf{CoSieve}(v_i, r') \bullet \mathsf{Sieve}(v_j, s')$ inherently contains all paths present in $\mathsf{CoSieve}(v_i, r) \bullet \mathsf{Sieve}(v_j, s)$. This guarantees that increasing the sieve radii monotonically increases the topological distinguishing power of the initial tuple hashing.

Without simulating higher-order k-tuples, SNN leverages the expanded topological pathways of $\mathsf{CoSieve}(v_i, r) \bullet \mathsf{Sieve}(v_j, s)$ to natively capture structural information. Empirically, this leverage is evaluated by SNN achieving a 0% failure rate on SRG families and the BREC benchmark (Table 1), cleanly distinguishing non-isomorphic structures that bounded k-WL models cannot.

5.4 Further examples

To illustrate the flexibility of the GkGNN framework in incorporating diverse structural principles—deterministic or stochastic—into unified message-passing strategies, we present two examples that are entirely distinct from the sieve-based construction of SNN.

Example 5.4.1. ***Random Walk–Based Monoidal Elements.***

Let $G = (V, E)$ *be a graph. We construct stochastic elements of* $\mathsf{SMult}(G)$ *via a simple random-walk mechanism.*

Step 1. *For each node* $v \in V$*, independently and uniformly select a neighbor* $w \in N(v)$ *and create a directed edge* $(v \to w)$*.*

Step 2. *Let* G_i *be the directed multigraph consisting of all such edges. We equip* G_i *with the set of its directed edges, viewed as length-one paths, and define*

$$\hat{G}_i := \big(G_i, E(G_i)\big) \in \mathsf{SMult}(G).$$

Repeating this procedure for t independent iterations yields elements $\hat{G}_1, \ldots, \hat{G}_t \in \mathsf{SMult}(G)$. Their monoidal composition

$$\hat{G}_1 \bullet \hat{G}_2 \bullet \cdots \bullet \hat{G}_t$$

encodes multi-step transition structure induced by successive random walks. While each G_i consists only of one-step transitions, longer paths arise through the composition operation $\bullet$.

To implement this construction, let $A_1, \ldots, A_t$ be the adjacency matrices of the directed multigraphs $G_1, \ldots, G_t$. Applying the GkGNN matrix operation yields

$$A_1 \circ A_2 \circ \cdots \circ A_t \in \mathsf{Mom}(G).$$

The resulting matrix aggregates stochastic walk information and defines a message-passing operator driven by randomized structural exploration.

Example 5.4.2. ***Flow-Based Construction via Partitions.***

Let $G = (V, E)$ be a graph and suppose that $\{V_i\}_{i=1}^k$ is a partition of V, i.e.,

$$\bigcup_{i=1}^{k} V_i = V, \qquad V_i \cap V_j = \varnothing \quad \text{for } i \neq j.$$

For each block V_i, define

$$\mathsf{Mask}(V_i) := \bigoplus_{v_j \in V_i} \mathsf{Tr}(N(v_j))^\top.$$

Operationally, $\mathsf{Mask}(V_i)^\top$ is obtained from the adjacency matrix by zeroing out all rows corresponding to nodes not in V_i. We then define

$$F = \mathsf{Mask}(V_1) \circ \mathsf{Mask}(V_2) \circ \cdots \circ \mathsf{Mask}(V_k) \in \mathsf{Mat}_{|V|}(\mathbb{R}).$$

The operator F encodes ordered information flow across partition blocks, from V_1 through V_k, and thus defines a structured propagation mechanism determined by the chosen partition. In particular, contributions from nodes in V_i can influence nodes in V_j only through compositions consistent with the block order, thereby enforcing a coarse-grained flow of information across the graph.

Since the partition of V may be defined arbitrarily (e.g., based on node degree, community structure, or a probability distribution over nodes), this construction illustrates how external structural criteria can be algebraically incorporated into message passing within the GkGNN framework. In this sense, partitions of the node set naturally induce families of propagation operators, demonstrating that GkGNN can integrate clustering or community-level structure directly into the design of message-passing mechanisms.

6 Experiments

To show how a shift in perspective improves graph understanding, we conduct a comprehensive evaluation of SNN on two tasks: (i) graph isomorphism and (ii) a topology-encoding probe.

6.1 Graph isomorphism.

We evaluate SNN on standard isomorphism benchmarks to test its ability to distinguish non-isomorphic graphs and to compare against WL-style limits. Throughout, we use the β variants in strong (saturated) settings, and we do *not* train any parameters: each graph G is mapped once to an SNN output matrix $S(G)$, from which we compute simple permutation-invariant summaries as embeddings. By invariance, isomorphic graphs yield identical embeddings; non-identical embeddings imply non-isomorphism.[2]

[2]In practice we compare embeddings with a small numerical tolerance. The depth "-1" denotes the saturated sieve (Section 5).

Table 1: Left: Failure rates of 3-WL and SNN across Strongly Regular graphs. Right: Number of distinguished pairs on **BREC**. Baseline values from Wang & Zhang (2024).

Graph Category	**3-WL (%)**	**SNN (%)**	**Graph Category**	**3-WL (%)**	**SNN (%)**
SRG(25,12,5,6)	100	0	SRG(36,15,6,6)	100	0
SRG(26,10,3,4)	100	0	SRG(37,18,8,9)	100	0
SRG(28,12,6,4)	100	0	SRG(40,12,2,4)	100	0
SRG(29,14,6,7)	100	0	SRG(65,32,15,16)	100	0
SRG(35,16,6,8)	100	0			
SRG(35,18,9,9)	100	0			
SRG(36,14,4,6)	100	0			

Model	Basic (60)	Reg. (140)	Ext. (100)	CFI (100)
3-WL	60	50	100	60
SSWL-P	60	50	100	38
I^2-GNN	60	100	100	21
GSN	60	99	95	0
PPGN	60	50	100	23
SNN	60	140	100	100

SR (Strongly Regular graphs). We use **all** publicly available collections of strongly regular graphs from Brendan McKay's Graph Data (archived). SR graphs are challenging since 3-WL cannot fully distinguish them (Bodnar et al., 2021b). For each collection, we apply $\mathsf{SNN}(\beta, (-1, -1, -1))$ to every graph G and compute a 6-dimensional embedding from $S(G)$: $\big(\det(S), \mathrm{Min}(S), \mathrm{Mean}(S),\ \mathrm{Var}(S),\ \mathrm{Mean}(\mathrm{diag}(S)),\ \mathrm{Var}(\mathrm{diag}(S))\big)$. By invariance and Theorem 4.1.2, $\det(S)$ is preserved under relabeling, so isomorphic graphs match. Within each collection, SNN assigns distinct embeddings to *all* graphs, yielding a 0% failure rate; see Table 1.

CSL (Circular Skip Links). CSL contains 150 4-regular graphs partitioned into 10 isomorphism classes and is widely used to probe GNN expressivity (Murphy et al., 2019; Dwivedi et al., 2023). We run $\mathsf{SNN}(\beta, (-1))$ on each graph and use $\mathrm{Sum}(S)$ (sum of all entries of S) as a permutation-invariant scalar embedding. The resulting values perfectly separate the 10 classes: graphs within a class share the same value; graphs from different classes do not.

BREC. BREC (Wang & Zhang, 2024) contains 400 pairs of non-isomorphic graphs divided into four categories (60 Basic, 140 Regular, 100 Extension, and 100 CFI), with cases that remain indistinguishable even under the 4-WL test. For each graph G, we apply $\mathsf{SNN}(\beta, (-1, -1, -1, -1))$ and construct the same type of embedding as used in the **SR** experiment. Across all BREC pairs, SNN consistently assigns distinct embeddings to the two graphs in each pair, yielding a 0% failure rate (Table 1).

6.2 Topology Encoding (probe).

Our aim here is to evaluate *only* the topology encoded by SNN as a preprocessing operator, independently of any learnable parameters or downstream training. To do so, we design a parameter-free *probe* based on one-step Label Propagation (LP) (Zhu & Ghahramani, 2002; Huang et al., 2020) with $\alpha = 1$. This choice isolates the structural signal present in the propagation operator and avoids confounds from optimization, regularization, or model capacity.

We compare two families of propagation operators. The first consists of classical adjacency-based operators, including the adjacency matrix Ad and polynomial variants such as Ad^2, $Ad + Ad^2$, and $Ad + Ad^2 + Ad^3$. The second family consists of operators generated by SNN. Concretely, we run LP on $A \in \{Ad,\ Ad^2,\ Ad + Ad^2,\ Ad + Ad^2 + Ad^3,\ \mathsf{SNN}(\beta, (1, 1)),\ \mathsf{SNN}(\beta, (1, 1, 1)),\ \mathsf{SNN}(\beta, (1, 2)),\ \mathsf{SNN}(\beta, (2, 2))\}$.

LP update (one step, no learning). Given initial one-hot labels $Y^{(0)}$ on the training nodes (zeros elsewhere), we perform a single propagation step $Y^{(1)} = \widehat{M}Y^{(0)}$. Here $\widehat{M}$ is a normalized version of the chosen operator A. We report results for three standard normalizations $\widehat{M} \in \left\{D^{-1/2}AD^{-1/2},\ D^{-1}A,\ AD^{-1}\right\}$, denoted respectively as `DAD`, `DA`, and `AD`, where D is the degree matrix induced by A. This ensures comparability across covers and conforms to standard LP practice. We evaluate on **Cora**, **CiteSeer**, **PubMed**, **ogbn-arxiv**, **AmazonPhoto**, and **Actor**; the dataset specifications and the runtimes of models $\mathsf{SNN}(\beta, (1, 1))$ and $\mathsf{SNN}(\beta, (1, 1, 1))$ are reported in Table 4. Because LP has no learnable parameters, differences in accuracy primarily reflect the structural information encoded by the propagation operator A.

Table 2: Test accuracy of one-step Label Propagation ($\alpha = 1$) using different propagation operators. The upper part reports classical polynomial adjacency operators, while the lower part reports operators derived from SNN. Since the probe uses a single propagation step and no learnable parameters, differences in accuracy reflect the structural information encoded by the propagation operator. The best classical baseline is underlined and the best overall result is shown in bold.

	Ad			Ad^2			$Ad + Ad^2$			$Ad + Ad^2 + Ad^3$		
Dataset	DAD	DA	AD	DAD	DA	AD	DAD	DA	AD	DAD	DA	AD
Cora	0.260	0.260	0.258	0.485	0.483	0.477	0.514	0.508	0.506	<u>0.607</u>	0.600	0.600
CiteSeer	0.137	0.137	0.137	0.229	0.225	0.227	0.257	0.254	0.254	<u>0.377</u>	0.368	0.373
PubMed	0.189	0.189	0.189	0.242	0.242	0.240	0.248	0.248	0.246	0.423	<u>0.428</u>	0.422
ogbn-arxiv	0.617	0.597	0.613	0.634	0.613	0.595	<u>0.660</u>	0.640	0.619	0.637	0.636	0.567
AmazonPhoto	0.337	0.337	0.344	0.742	0.738	0.736	0.742	0.738	0.739	**<u>0.827</u>**	0.792	0.761
Actor	0.195	0.196	0.191	0.193	0.188	0.191	0.191	0.190	0.190	0.190	<u>0.195</u>	0.191

	$\mathsf{SNN}(\beta,(1,1))$			$\mathsf{SNN}(\beta,(1,2))$			$\mathsf{SNN}(\beta,(1,1,1))$			$\mathsf{SNN}(\beta,(2,2))$		
Dataset	DAD	DA	AD	DAD	DA	AD	DAD	DA	AD	DAD	DA	AD
Cora	0.512	0.505	0.507	0.608	0.599	0.608	0.609	0.602	0.608	**0.669**	0.647	0.624
CiteSeer	0.261	0.258	0.259	0.368	0.369	0.369	0.370	0.368	0.372	0.427	0.416	**0.438**
PubMed	0.248	0.248	0.246	0.423	0.429	0.425	0.423	0.429	0.424	0.638	**0.642**	0.621
ogbn-arxiv	**0.663**	0.645	0.625	0.649	0.645	0.621	0.647	0.642	0.581	0.641	0.642	0.617
AmazonPhoto	0.743	0.739	0.741	0.817	0.789	0.770	0.826	0.796	0.766	0.808	0.774	0.702
Actor	0.188	0.192	0.185	0.253	**0.254**	0.245	0.190	0.191	0.191	0.254	0.251	0.241

Discussion. Table 2 compares classical polynomial propagation operators with operators derived from sieve covers under the same one-step label propagation probe. Because the probe uses a single propagation step and no learnable parameters, differences in accuracy reflect only the structural information encoded by the propagation matrix.

Among classical operators, increasing the degree of the adjacency polynomial often improves performance on several citation-style datasets. For example, on Cora and CiteSeer, moving from the adjacency matrix to Ad^2, $Ad + Ad^2$, and $Ad + Ad^2 + Ad^3$ gradually increases accuracy, suggesting that incorporating longer walks allows the operator to capture broader connectivity patterns.

However, increasing polynomial depth does not always lead to further improvements. On *ogbn-arxiv*, the best classical operator is $Ad + Ad^2$ (0.660), while adding the third power of Ad slightly decreases performance. This suggests that simply increasing the propagation radius does not necessarily provide additional useful structural information on larger graphs.

The Actor dataset provides a particularly clear contrast between the two operator families. Classical polynomial operators remain close to the adjacency baseline (around 0.19), whereas sieve-based operators reach 0.254. Since the probe does not involve training, this improvement reflects differences in the topology encoded by the propagation operators themselves. The results on Actor also indicate that the specific structure of the sieve can influence performance. For instance, $\mathsf{SNN}(\beta,(1,2))$ achieves higher accuracy than $\mathsf{SNN}(\beta,(1,1,1))$.

Overall, these results illustrate that polynomial adjacency operators can capture useful multi-hop connectivity patterns, while operators derived from sieve covers may encode additional structural information in certain graphs. In particular, the Actor dataset highlights a setting where this difference becomes visible even under a single propagation step.

6.3 Graph classification

We evaluate SNN on four datasets from the TUD benchmark suite: **NCI1**, **IMDB-B**, **IMDB-M**, and **PROTEINS**. Our model is compared against several GNNs, including GIN Xu et al. (2019), PPGNs Maron et al. (2019), GSN Bouritsas et al. (2023), TL-GNN Ai et al. (2022), SIN Bodnar et al. (2021b),

Table 3: Accuracy on TUD datasets. The best result for each dataset is shown in bold and the second best is underlined. ⋆Graph Kernel Methods

Dataset	NCI1	IMDB-B	IMDB-M	PROTEINS
WL kernel⋆ Shervashidze et al. (2011)	**86.0±1.8**	73.8±3.9	50.9±3.8	75.0±3.1
GNTK⋆ Du et al. (2019)	<u>84.2±1.5</u>	76.9±3.6	52.8±4.6	75.6±4.2
GIN Xu et al. (2019)	82.7±1.7	75.1±5.1	52.3±2.8	76.2±2.8
PPGNs Maron et al. (2019)	83.2±1.1	73.0±5.8	50.5±3.6	77.2±4.7
GSN Bouritsas et al. (2023)	83.5±2.0	77.8±3.3	54.3±3.3	76.6±5.0
TL-GNN Ai et al. (2022)	83.0±2.1	<u>79.7±1.9</u>	**55.1±3.2**	**79.9±4.4**
SIN Bodnar et al. (2021b)	82.8±2.2	75.6±3.2	52.5±3.0	76.5±3.4
CIN Bodnar et al. (2021a)	83.6±1.4	75.6±3.7	52.7±3.1	<u>77.0±4.3</u>
SNN	83.6±1.2	**80.5±3.0**	<u>54.53±2.23</u>	<u>78.70±4.29</u>

and CIN Bodnar et al. (2021a), as well as classical graph kernels such as the WL kernel Shervashidze et al. (2011) and GNTK Du et al. (2019). For all datasets except **NCI1**, we employ $\mathsf{SNN}(\alpha, (1, 1))$. Recognizing the need for a more expressive variant for **NCI1**, we utilize $\mathsf{SNN}(\alpha, (1, 2))$.

Equation 3 defines the propagation operator through the matrices $\mathsf{Tr}(D_i(v))$. Multiplying $\mathsf{Tr}(D_i(v))$ by a constant $\gamma \in (0, 1]$ enhances the model in a manner that is sensitive to the length of paths. In our experiments, we set $\gamma = 0.5$ in all cases. For other hyperparameters, see Table 5. We treat the output of SNN as a weighted graph that is subsequently processed by a standard GNN layer. We utilize the GNN operator **GraphConv** provided by PyTorch Geometric (Fey & Lenssen, 2019), based on the model introduced in (Morris et al., 2019). Tenfold cross-validation is performed. The results, presented in Table 3, show that SNN achieves good performance across this diverse set of datasets.

Remarks on heterophilic graphs. Many real-world graphs exhibit heterophily, where neighboring nodes often belong to different classes. In such settings, propagation based purely on immediate neighborhoods may be less informative. The GkGNN framework allows propagation operators to be constructed from structured compositions of paths rather than relying solely on adjacency. In particular, SNN aggregates information along direction-respecting paths induced by the chosen cover. For illustration, we report results on the Actor dataset in Table 2, which is commonly regarded as heterophilic. In this experiment we apply one-step label propagation ($\alpha = 1$) on operators derived from SNN without any training. Using the operator $\mathrm{SNN}(\beta, (1, 2))$ as a preprocessing transformation yields 25.4% accuracy. For reference, the performance of a trained GCN baseline on this dataset is reported as 26.86% in Pei et al. (2020). More generally, the GkGNN framework allows propagation operators to incorporate external structural criteria. For example, Example 5.4.2 illustrates how a partition of the node set can induce a structured flow operator that constrains how information propagates across groups of nodes.

Limitations. GkGNN is a general design framework rather than a single model, with an intentionally broad design space. Consequently, practical limitations depend on how the framework is instantiated through specific cover constructions. In this work, we therefore discuss limitations primarily at the level of SNN, which serves as an instantiation. In our experiments, we focus on SNN. We observe that highly saturated variants of SNN, while effective at exposing structural differences between non-isomorphic graphs, may be overly discriminative for certain graph classification tasks. In particular, saturation can amplify fine-grained structural distinctions to the extent that similar graphs within the same class become less aligned in the induced representations, which may reduce classification robustness in downstream tasks. Another practical limitation is that saturated sieve constructions often produce dense matrices. While this is acceptable for offline analysis and expressivity probes, practical deployment may require sparsification or masking strategies to control computational and memory costs. As discussed in Subsection 6.3, the output of SNN can be enhanced to become sensitive to the length of paths. In such settings, graph sparsification techniques could potentially be applied to control the density of the resulting operators. For example, spectral sparsification methods such as Spielman & Srivastava (2011) aim to approximate dense graphs with sparser ones while preserving important structural properties. More broadly, a range of graph reduction techniques—including sparsification, coarsening, and condensation—have been studied to control computational costs in graph

learning pipelines Hashemi et al. (2024). In practice, sparsification may be applied at different stages of the pipeline. For graphs that are already dense, it may be preferable to first sparsify the graph before constructing the SNN operator, in order to prevent excessive density in the resulting matrices. Recent learning-based sparsification approaches also explore constructing task-adaptive sparse graphs for scalable GNN computation Zhang et al. (2025). In addition, stochastic edge-thinning approaches such as *DropEdge* Rong et al. (2020), which randomly remove edges during training, might also be considered when SNN operators are used within learning pipelines. Exploring such strategies within the GkGNN framework is left for future work.

7 Conclusion

We formalized *covers* as a strict algebraic extension of neighborhoods and, in doing so, established a new foundation for message passing in which neighborhoods are recovered as a special case. We introduced the GkGNN framework to systematically construct covers and translate them into matrices, providing a principled mechanism for designing message-passing operators beyond adjacency-based aggregation. This platform simplifies the development of topology-aware propagation schemes. As a concrete instantiation, we proposed Sieve Neural Networks (SNN), which operationalize the framework and demonstrate strong performance on graph isomorphism and topology-encoding probes. This work focuses on the foundational layer of GkGNN. Looking ahead, we will deepen the theoretical analysis of GkGNN's expressive power and study how covers can be parameterized, learned, and adapted within end-to-end architectures. We also plan to broaden the scope of applications, including a more comprehensive theoretical and empirical comparison between SNN, GkGNN-based models, and the Weisfeiler–Lehman hierarchy.

References

Xing Ai, Chengyu Sun, Zhihong Zhang, and Edwin R. Hancock. Two-level graph neural network. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–14, 2022. doi: 10.1109/TNNLS.2022.3144343.

Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Liò, Guido F Montufar, and Michael Bronstein. Weisfeiler and lehman go cellular: Cw networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 2625–2640. Curran Associates, Inc., 2021a. URL `https://proceedings.neurips.cc/paper_files/paper/2021/file/157792e4abb490f99dbd738483e0d2d4-Paper.pdf`.

Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lió, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 1026–1037. PMLR, 18–24 Jul 2021b. URL `https://proceedings.mlr.press/v139/bodnar21a.html`.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M. Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2023. doi: 10.1109/TPAMI.2022.3154319.

Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper_files/paper/2019/file/663fd3c5144fd10bd5ca6611a9a5b92d-Paper.pdf`.

Vijay Prakash Dwivedi, Chaitanya K. Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023. URL `http://jmlr.org/papers/v24/22-0567.html`.

Aosong Feng, Chenyu You, Shiqiang Wang, and Leandros Tassiulas. Kergnns: Interpretable graph neural networks with graph kernels. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6): 6614–6622, Jun. 2022a. doi: 10.1609/aaai.v36i6.20615. URL https://ojs.aaai.org/index.php/AAAI/article/view/20615.

Jiarui Feng, Yixin Chen, Fuhai Li, Anindya Sarkar, and Muhan Zhang. How powerful are k-hop message passing graph neural networks. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 4776–4790. Curran Associates, Inc., 2022b. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/1ece70d2259b8e9510e2d4ca8754cecf-Paper-Conference.pdf.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Johannes Gasteiger, Chandan Yeshwanth, and Stephan Günnemann. Directional message passing on molecular graphs via synthetic coordinates. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 15421–15433. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/82489c9737cc245530c7a6ebef3753ec-Paper.pdf.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1263–1272. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/gilmer17a.html.

Mustafa Hajij, Ghada Zamzmi, Theodore Papamarkou, Nina Miolane, Aldo Guzmán-Sáenz, Karthikeyan Natesan Ramamurthy, Tolga Birdal, Tamal K. Dey, Soham Mukherjee, Shreyas N. Samaga, Neal Livesay, Robin Walters, Paul Rosen, and Michael T. Schaub. Topological deep learning: Going beyond graph data, 2023. URL https://arxiv.org/abs/2206.00606.

Mohammad Hashemi, Shengbo Gong, Juntong Ni, Wenqi Fan, B. Aditya Prakash, and Wei Jin. A comprehensive survey on graph reduction: sparsification, coarsening, and condensation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, IJCAI '24, 2024. ISBN 978-1-956792-04-1. doi: 10.24963/ijcai.2024/891. URL https://doi.org/10.24963/ijcai.2024/891.

Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks, 2020. URL https://arxiv.org/abs/2010.13993.

Thomas W. Hungerford. *Algebra*. Springer New York, NY, 1980.

Saunders MacLane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory*. Universitext. Springer, 1994. doi: 10.1007/978-1-4612-0927-0.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bb04af0f7ecaee4aae62035497da1387-Paper.pdf.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4602–4609, Jul. 2019. doi: 10.1609/aaai.v33i01.33014602. URL https://ojs.aaai.org/index.php/AAAI/article/view/4384.

Ryan Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Relational pooling for graph representations. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4663–4673. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/murphy19a.html.

Mathilde Papillon, Guillermo Bernárdez, Claudio Battiloro, and Nina Miolane. Topotune : A framework for generalized combinatorial complex neural networks, 2025. URL https://arxiv.org/abs/2410.06530.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 21997–22009. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/b8b2926bd27d4307569ad119b6025f94-Paper.pdf.

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 (ICLR)*, 2020.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR*, 2020.

Ryoma Sato. A survey on the expressive power of graph neural networks. *arXiv preprint arXiv:2003.04078*, 2020.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011. URL http://jmlr.org/papers/v12/shervashidze11a.html.

Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *SIAM Journal on Computing*, 2011.

Clément Vignac, Andreas Loukas, and Pascal Frossard. Building powerful and equivariant graph neural networks with structural message-passing. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 14143–14155. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/a32d7eeaae19821fd9ce317f3ce952a7-Paper.pdf.

Yanbo Wang and Muhan Zhang. An empirical study of realized GNN expressiveness. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 52134–52155. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/wang24cl.html.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10737–10745, May 2021. doi: 10.1609/aaai.v35i12.17283. URL https://ojs.aaai.org/index.php/AAAI/article/view/17283.

Guibin Zhang, Xiangguo SUN, Yanwei Yue, Chonghe Jiang, Kun Wang, Tianlong Chen, and Shirui Pan. Graph sparsification via mixture of graphs. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 92735–92763, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/e7947b5e1d30864ebbe8714dbdd611d9-Paper-Conference.pdf.

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical Report CMU-CALD-02-107, Carnegie Mellon University, 2002.

A Definitions and examples

A.1 Definitions

The definition of a monoid and monoidal homomorphism are as follows (Hungerford, 1980):

Definition A.1.1. *A monoid is a non-empty set* M *together with a binary operation* $\cdot$ *on* M *which*

1) is associative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ *for all* $a, b, c \in \mathsf{M}$ *and*

2) contains identity element $e \in \mathsf{M}$ *such that* $a \cdot e = e \cdot a = a$

If, for all $a, b \in \mathsf{M}$*, the operation satisfies* $a \cdot b = b \cdot a$*, then we say that* M *is a commutative monoid.*

Definition A.1.2. *A* monoid homomorphism *between monoids* $(M, \bullet)$ *and* $(N, \circ)$ *with identity elements* e_M *and* e_N*, respectively, is a function* $f : M \to N$ *such that*

$$f(x \bullet y) \;=\; f(x) \circ f(y) \quad \textit{for all } x, y \in M, \qquad f(e_M) = e_N.$$

A.2 Examples

Example A.2.1. *Considering a Change-of-Order mapping* $f : \mathsf{Mat}_3(\mathbb{R}) \to \mathsf{Mat}_3(\mathbb{R})$*, obtained by reordering the standard basis* $\{e_1, e_2, e_3\}$ *to the basis* $\{e_3, e_2, e_1\}$*. For a given matrix* A*, we get the matrix* $f(A)$ *as follows:*

$$A \longmapsto f(A)$$

$$\begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \end{array} \begin{array}{c} \begin{array}{ccc} e_1 & e_2 & e_3 \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \end{array} \xmapsto{f:e_1 \leftrightarrow e_3} \begin{array}{c} \\ e_3 \\ e_2 \\ e_1 \end{array} \begin{array}{c} \begin{array}{ccc} e_3 & e_2 & e_1 \end{array} \\ \begin{pmatrix} a_{33} & a_{32} & a_{31} \\ a_{23} & a_{22} & a_{21} \\ a_{13} & a_{12} & a_{11} \end{pmatrix} \end{array}$$

B Explanation for constructing a model in GkGNN framework

The process of designing a GNN model within this framework is outlined as follows:

1) For a given graph G, the process involves selecting a collection $\mathcal{C}_G$ of elements from $\mathsf{Mod}(G)$ to serve as a cover for G. These elements can be generated using $\mathsf{DirSub}(G)$ and the binary operation $\bullet$. Notably, Theorem 3.2.5 ensures the ability to create any suitable and desired elements by leveraging directed edges and the operator $\bullet$.

2) Next, the chosen cover is transformed into a collection of matrices within $\mathsf{Mom}(G)$, utilizing Tr. During this transformation, the operation $\circ$ and other elements of $\mathsf{Mom}(G)$ can be employed to convert the original collection into a new one. The resulting output at this stage is denoted by $\mathcal{A}_G$.

3) By utilizing ι, the collection obtained in the second stage transitions into a larger and more equipped space, a suitable environment for enrichment. This stage leverages all the operations outlined in Proposition 4.1.1 to complete the model's design. Following the processing of $\mathcal{A}_G$ in this stage, we obtain a new collection of matrices denoted by $\mathcal{M}_G$, representing the model's output.

Hence, a model is a mapping that associates a collection of matrices $\mathcal{M}_G$ with a given graph G. $\mathcal{M}_G$ plays a role akin to the adjacency matrix and provides an interpretation of the chosen cover for use in various forms of message passing. While the second and third stages can be merged, we prefer to emphasize the significance of Tr in this process.

This construction of a model is appropriate for tasks such as node classification. For graph classification, we need an invariant construction. Based on Theorem 4.1.2, a graph isomorphism $f : G \to H$ transform the triple $(\mathcal{C}_G, \mathcal{A}_G, \mathcal{M}_G)$ to a triple $(\mathcal{C}'_H, \mathcal{A}'_H, \mathcal{M}'_H)$ for graph H and this may be different from $(\mathcal{C}_H, \mathcal{A}_H, \mathcal{M}_H)$. So a model constructed in the GkGNN framework is invariant if for every graph isomorphism $f : G \to H$, the maps $\mathsf{Mod}(f)$, $\mathsf{Mom}(f)$ and $\mathsf{CO}(f)$ induce one-to-one correspondences between $\mathcal{C}_G$ and $\mathcal{C}_H$, $\mathcal{A}_G$ and $\mathcal{A}_H$, and $\mathcal{M}_G$ and $\mathcal{M}_H$, respectively. The model SNN is an example of an invariant model.

C GkGNN framework vs. higher-order GNNs: a comparison

We contrast GkGNN with higher-order GNNs such as MPSN Bodnar et al. (2021b), CWN Bodnar et al. (2021a), GSN Bouritsas et al. (2023), and TLGNN Ai et al. (2022).

Framework, not a single model. GkGNN is a *design framework*: it gives precise, graph-agnostic definitions of *covers* (generalizing neighborhoods) and a principled way to turn them into matrices. Whereas higher-order GNNs typically hard-code one specific alternative to neighborhood aggregation, GkGNN provides an *infinite design space* of covers, of which the standard neighborhood cover is a special case, enabling diverse message-passing strategies tailored to a task.

Topology-aware by construction. By Theorems 4.1.2 and 4.1.3, GkGNN yields an algebraic description of a graph that is unique up to isomorphism. Each monoidal element of $\mathsf{Mod}(G)$ encodes concrete topological relationships; choosing a cover selects which aspects of topology to expose to downstream GNNs. Moreover, the algebra (composition, translation to matrices) lets one combine ideas from other paradigms within a single coherent toolkit.

Example: recovering k-hop message passing. GkGNN can reproduce common higher-order behaviors. Starting from the neighborhood cover $\{S_v : v \in G\}$, define for a node v_k the set

$$\text{2-}\mathsf{hop}(v_k) = \left\{ S_{v_{k_i}} \ \bullet \ e_i \ : \ v_{k_i} \in N(v_k), \ e_i : v_{k_i} \to v_k \right\}.$$

Let $\text{2-}\mathsf{hop}(G) = \bigcup_{v_k} \text{2-}\mathsf{hop}(v_k)$. Applying Tr maps this cover to a collection of matrices, which can be aggregated (e.g., by summation) to obtain a 2-hop propagation operator, mirroring the effect of k-hop message passing in Feng et al. (2022b).

D Further details on SNN

D.1 Model explanations

The SNN construction provides two ways to collapse the matrix collection induced by the sieve cover into a single operator: the α- and β-variants.

α-variant. Using $\mathsf{CoImage}(v, l) = \mathsf{Image}(v, l)^\top$, we obtain

$$\mathsf{CoImage}(v_i, l) \circ \mathsf{Image}(v_j, k) = \big(\mathsf{CoImage}(v_j, k) \circ \mathsf{Image}(v_i, l) \big)^\top.$$

Hence the output of $\mathsf{SNN}(\alpha, (l, k))$ is the transpose of the output of $\mathsf{SNN}(\alpha, (k, l))$, and $\mathsf{SNN}(\alpha, (l, l))$ is symmetric. For $l \neq k$, symmetry need not hold (cf. Example D.2.1), so $\mathsf{SNN}(\alpha, (l, k))$ and $\mathsf{SNN}(\alpha, (k, l))$ may differ. Moreover, increasing the radii only adds admissible paths: if $l \leq l'$ and $k \leq k'$, then $\mathsf{SNN}(\alpha, (l', k'))$ captures (entrywise) at least as many paths as $\mathsf{SNN}(\alpha, (l, k))$.

β-variant. The families $\{\mathsf{Sieve}(v, l_i)\}_v$ (or $\{\mathsf{CoSieve}(v, l_i)\}_v$) form subcovers of the cover of sieves. Their matrix summaries

$$Su_i \ = \ \sum_{v \in V} \mathsf{Image}(v, l_i) \quad \text{or} \quad Su_i \ = \ \sum_{v \in V} \mathsf{CoImage}(v, l_i)$$

aggregate all allowed paths contributed by the chosen subcover. Composing these summaries with the monoid operation $\circ$ produces

$$Su_1 \circ \cdots \circ Su_t,$$

which realizes a specific combination of subcovers: paths admitted by earlier subcovers are composed with those of later ones. Because $\circ$ is, in general, noncommutative, the order of Su_i reflects the intended sequencing of interactions encoded by the cover.

D.2 Comparing with MPNN

For a node v, its neighborhood can be described by the element $\mathsf{Sieve}(v, 1)$. Consequently, $\mathsf{SNN}_o(\alpha, (0, 1))$ and $\mathsf{SNN}_o(\alpha, (1, 0))$ correspond to the adjacency matrix, signifying their utilization of neighborhoods for message passing. This is equivalent to MPNNs. Hence, SNN can be considered as a generalization of MPNNs. In the following example, two graphs are considered that MPNN can not distinguish, yet SNN can. This example illustrates how a shift in perspective, resulting from a change in cover, reveals the topological properties of graphs.

Example D.2.1. *The graphs in Figure 4 are not distinguishable by MPNN (Sato, 2020) because they are locally the same. Applying* $\mathsf{SNN}_o(\alpha, (1, 1))$*, a level of version* α *of* SNN *that is slightly more potent than*

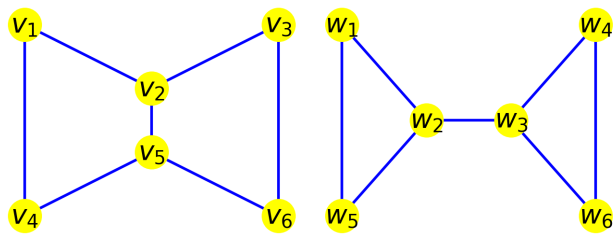

Figure 4: The graph G, the left one, and H, the right one, are not distinguishable by MPNN

MPNN, we get the following symmetric matrices X *and* Y *for* G *and* H *respectively as the outputs of the model for these graphs.*

$$X = \begin{pmatrix} 2 & 2 & 1 & 2 & 2 & 0 \\ 2 & 3 & 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 0 & 2 & 2 \\ 2 & 2 & 0 & 2 & 2 & 1 \\ 2 & 2 & 2 & 2 & 3 & 2 \\ 0 & 2 & 2 & 1 & 2 & 2 \end{pmatrix} Y = \begin{pmatrix} 2 & 3 & 1 & 0 & 3 & 0 \\ 3 & 3 & 2 & 1 & 3 & 1 \\ 1 & 2 & 3 & 3 & 1 & 3 \\ 0 & 1 & 3 & 2 & 0 & 3 \\ 3 & 3 & 1 & 0 & 2 & 0 \\ 0 & 1 & 3 & 3 & 0 & 2 \end{pmatrix}$$

The entry ij *in these matrices corresponds to the count of paths between nodes* v_i *and* v_j *in* $\mathsf{CoSieve}(v_i, 1) \bullet \mathsf{Sieve}(v_j, 1)$ *and* w_i *and* w_j *in* $\mathsf{CoSieve}(w_i, 1) \bullet \mathsf{Sieve}(w_j, 1)$*. The disparity between these matrices highlights the differences between the graphs. This dissimilarity becomes more apparent when applying the set function* Var*, while* Sum *and* Mean *yield identical values. When* $\mathsf{SNN}_o(\alpha, (1, 2))$*, a more complex level of* SNN*, is applied, we obtain the following nonsymmetric matrices, denoted as* Z *and* W*, for graphs* G *and* H*. Applying all three set functions results in distinct outputs, further emphasizing the dissimilarity between the graphs.*

$$Z = \begin{pmatrix} 2 & 4 & 2 & 4 & 4 & 3 \\ 5 & 3 & 5 & 4 & 6 & 4 \\ 2 & 4 & 2 & 3 & 4 & 4 \\ 4 & 4 & 3 & 2 & 4 & 2 \\ 4 & 6 & 4 & 5 & 3 & 5 \\ 3 & 4 & 4 & 2 & 4 & 2 \end{pmatrix} W = \begin{pmatrix} 2 & 3 & 3 & 1 & 3 & 1 \\ 4 & 3 & 4 & 2 & 4 & 2 \\ 2 & 4 & 3 & 4 & 2 & 4 \\ 1 & 3 & 3 & 2 & 1 & 3 \\ 3 & 3 & 3 & 1 & 2 & 1 \\ 1 & 3 & 3 & 3 & 1 & 2 \end{pmatrix}$$

D.3 Complexity

SNN is applied *once* as a preprocessing step to convert each input graph (or a dataset of graphs) into its transformed counterpart; it is not used during training.

Let $G = (V, E)$ with $|V| = n$ and $|E| = m$. From Eq. equation 3, $\mathsf{Image}(v, k)$ is obtained by k iterations of adjacency-based additions/multiplications. The cost depends on the configuration:

- $\mathsf{SNN}(\beta, (1, \ldots, 1))$**.** In this case $\mathsf{Image}(v, k)$ can be read off directly from the adjacency matrix (no matrix–matrix products), so each Su_i equals the adjacency matrix. Hence computing $S^{(\beta)}$ is $\mathcal{O}(mn)$.

Table 4: Dataset statistics and runtime (in seconds) for constructing SNN-transformed graphs. Reported times correspond to building the propagation operator before applying Label Propagation.

Dataset	**#Nodes**	**#Edges**	**#Features**	**#Classes**	**Runtime** $\mathsf{SNN}(\beta,(1,1))$	**Runtime** $\mathsf{SNN}(\beta,(1,1,1))$
Cora	2,708	10,556	1,433	7	0.0525	0.0758
CiteSeer	3,327	9,104	3,703	6	0.0189	0.0319
PubMed	19,717	88,648	500	3	0.8055	1.3902
ogbn-arxiv	169,343	1,166,243	128	40	6.75	9.38
AmazonPhoto	7,650	238,162	745	8	0.6560	3.4766
Actor	7,600	30,019	932	5	0.0224	0.1543

- $\mathsf{SNN}(\alpha,(l,k))$ **or** $\mathsf{SNN}(\beta,(l_1,\ldots,l_t))$ **with** $k>1$ **or some** $l_{i_0}>1$**.** These require matrix-based compositions; computing $\mathsf{Image}(v,k)$ for a single node costs $\mathcal{O}(mn)$, yielding $\mathcal{O}(mn^2)$ over all nodes.

Since SNN runs only once to produce the transformed graphs, its runtime is incurred offline and does not affect the training-time complexity of downstream GNNs.

Table 5: Training hyperparameters used for each dataset. All experiments were conducted on Google Colab using only CPU and standard RAM.

Dataset	**Learning Rate**	**Batch Size**	**GNN Layers**	**Dropout**	**Hidden Channels**
NCI1	$1e^{-3}$	32	4	0.5	32
IMDB-B	$1e^{-3}$	32	4	0.5	64
IMDB-M	$1e^{-3}$	32	4	0.5	128
PROTEINS	$1e^{-3}$	32	4	0	32

D.4 Algorithm

Algorithm 1 Computing $\mathsf{Image}(v, k)$

Input: node v, integer k
Output: $\mathsf{Image}(v, k)$
Initialization: $N_0(v) = \{v\}$, $\mathsf{Image}(v, 0) = \textit{Zero matrix}$
for $i = 1, \cdots, k$ **do**
  $N_i(v) = \bigcup_{u \in N_{i-1}(v)} N(u) - \bigcup_{j=0}^{i-1} N_j(v)$
  $M_i(v) = \{w \to u : wu \in E, w \in N_i(v), u \in N_{i-1}(v)\}$
  $\mathsf{Tr}(D_i(v)) = \sum_{e \in M_i(v)} \mathsf{Tr}(e) =$ The adjacency matrix of directed subgraph $M_i(v)$
  $\mathsf{Image}(v, i) = \mathsf{Tr}(D_i(v)) \circ \mathsf{Image}(v, i-1)$
end for
Return: Final result

Algorithm 2 Computing $\mathsf{ColImage}(v, k)$

Input: $\mathsf{Image}(v, k)$
Output: $\mathsf{ColImage}(v, k)$
$\mathsf{ColImage}(\mathsf{v}, \mathsf{k}) =$ Transpose of $\mathsf{Image}(\mathsf{v}, \mathsf{k})$
Return: Final result

Algorithm 3 Computing $\mathsf{SNN}(\alpha, (l, k))$

Input: $\mathsf{Image}(v, k)$ and $\mathsf{ColImage}(v, l)$ for all $v \in V$
Output: $\mathsf{SNN}(\alpha, (l, k))$
Initialization: $\mathsf{SNN}(\alpha, (l, k)) = \textit{Zero matrix}$
for $v_i \in V$ **do**
  for $v_j \in V$ **do**
    $A = \mathsf{ColImage}(v_i, l) \circ \mathsf{Image}(v_j, k)$
    $r = \mathsf{ColImage}(v_i, l)[i, :].sum()$, summation of $i-$th row
    $c = \mathsf{Image}(v_j, k)[:, j].sum()$, summation of $j-$th column
    $\mathsf{SNN}(\alpha, (l, k))_{i,j} = \frac{A_{i,j}}{r \cdot c}$
  end for
end for
Return: Final result

Algorithm 4 Computing $\mathsf{SNN}(\beta, (l, k))$

Input: $\mathsf{Image}(v, k)$ and $\mathsf{ColImage}(v, l)$ for all $v \in V$
Output: $\mathsf{SNN}(\beta, (l, k))$
$Su_1 = \sum_{v \in V} \mathsf{ColImage}(v, l)$
$Su_2 = \sum_{v \in V} \mathsf{Image}(v, k)$
$\mathsf{SNN}(\beta, (l, k)) = Su_1 \circ Su_2$
Return: Final result

E Special submonoids

E.1 The submonoid generated by neighborhoods

The cover of neighborhoods, as a subset of $\mathsf{Mod}(G)$, generates a submonoid. To formalize this, let $\mathsf{Neigh}(G) \subseteq \mathsf{Mod}(G)$ and $\mathsf{Adj}(G) \subseteq \mathsf{Mom}(G)$ denote the submonoids generated by the cover of neighborhoods and its matrix transformation, respectively. The following theorems illustrate how these submonoids provide an algebraic characterization of a graph. It is straightforward to verify that for a graph isomorphism $f : G \to H$,

the mappings $\mathsf{Mod}(f)$ and $\mathsf{Mom}(f)$ send elements of $\mathsf{Neigh}(G)$ and $\mathsf{Adj}(G)$ to elements of $\mathsf{Neigh}(H)$ and $\mathsf{Adj}(H)$, respectively. Thus, as a consequence of Theorem 4.1.2, we have:

Theorem E.1.1. *Every graph isomorphism $f : G \to H$ induces monoidal isomorphisms $\mathsf{Neigh}(f) : \mathsf{Neigh}(G) \longrightarrow \mathsf{Neigh}(H)$ and $\mathsf{Adj}(f) : \mathsf{Adj}(G) \to \mathsf{Adj}(H)$ such that the following diagram is commutative, where ι represents the inclusions.*

$$\begin{tikzcd}
\mathsf{Neigh}(G) \arrow[r, "\mathsf{Tr}_G"] \arrow[d, "\mathsf{Neigh}(f)"'] & \mathsf{Adj}(G) \arrow[r, "\iota"] \arrow[d, "\mathsf{Adj}(f)"'] & \mathsf{Mat}_{|V_G|}(\mathbb{R}) \arrow[d, "\mathsf{CO}(f)"] \\
\mathsf{Neigh}(H) \arrow[r, "\mathsf{Tr}_H"'] & \mathsf{Adj}(H) \arrow[r, "\iota"'] & \mathsf{Mat}_{|V_H|}(\mathbb{R})
\end{tikzcd} \tag{4}$$

The converse of this theorem can be stated as follows:

Theorem E.1.2. *Suppose G and H are two graphs with $|V_G| = |V_H| = n$, and $f : \mathsf{Mat}_n(\mathbb{R}) \to \mathsf{Mat}_n(\mathbb{R})$ is a Change-of-Order mapping. If the restriction of f to $\mathsf{Adj}(G)$ yields an isomorphism to $\mathsf{Adj}(H)$, then G and H are isomorphic.*

Consequently, the horizontal homomorphisms in Diagram 4 can serve as an algebraic description of the graph. It demonstrates that the monoidal elements resulting from interactions between neighborhoods encapsulate richer information about the graph's topology. This suggests that the coverage of neighborhoods can be further enhanced by incorporating additional elements from $\mathsf{Neigh}(G)$.

E.2 The submonoid generated by sieves

The submonoid generated by the cover of Sieves fully determines the graph, as stated in the following two theorems. Let $\mathsf{Si}(G) \subseteq \mathsf{Mod}(G)$ and $\mathsf{Im}(G) \subseteq \mathsf{Mom}(G)$ denote the submonoids generated by the cover of sieves and its matrix transformation, respectively. As a direct consequence of Theorems 5.1.1 and 4.1.2, we have:

Theorem E.2.1. *Every graph isomorphism $f : G \to H$ induces monoidal isomorphisms $\mathsf{Si}(f) : \mathsf{Si}(G) \longrightarrow \mathsf{Si}(H)$ and $\mathsf{Im}(f) : \mathsf{Im}(G) \to \mathsf{Im}(H)$ such that the Diagram 5 is commutative, where ι represents the inclusions.*

$$\begin{tikzcd}
\mathsf{Si}(G) \arrow[r, "\mathsf{Tr}_G"] \arrow[d, "\mathsf{Si}(f)"'] & \mathsf{Im}(G) \arrow[r, "\iota"] \arrow[d, "\mathsf{Im}(f)"'] & \mathsf{Mat}_{|V_G|}(\mathbb{R}) \arrow[d, "\mathsf{CO}(f)"] \\
\mathsf{Si}(H) \arrow[r, "\mathsf{Tr}_H"'] & \mathsf{Im}(H) \arrow[r, "\iota"'] & \mathsf{Mat}_{|V_H|}(\mathbb{R})
\end{tikzcd} \tag{5}$$

The converse of the above theorem can be stated as follows:

Theorem E.2.2. *Suppose G and H are two graphs with $|V_G| = |V_H| = n$, and $f : \mathsf{Mat}_n(\mathbb{R}) \to \mathsf{Mat}_n(\mathbb{R})$ is a Change-of-Order mapping. If the restriction of f to $\mathsf{Im}(G)$ yields an isomorphism to $\mathsf{Im}(H)$, then G and H are isomorphic.*

The horizontal morphisms in Diagram 5 provide a unique, up-to-isomorphism algebraic characterization of a graph. This can be served as the basis for the significant performance of the model SNN in the graph isomorphism task, as demonstrated in the experimental section.

F Proof of theorems

F.1 Proof of Theorem 3.1.3

Proof. Since Rep is surjective, it suffices to demonstrate that Rep is also injective, meaning that if $\mathsf{Rep}(D) = \mathsf{Rep}(D')$, then $D = D'$. According to the matrix representation definition, $\leq_D = \leq_{D'}$. For an edge $v_i \xrightarrow{e} v_j$

in D, it implies $v_i \leq_D v_j$, and consequently, $v_i \leq_{D'} v_j$. Suppose $v_i \xrightarrow{e} v_j$ is not a directed edge in D'. In that case, there must be a path in D' traversing a node v_k different from v_i and v_j. This implies $v_i \leq_{D'} v_k$ and $v_k \leq_{D'} v_j$, and consequently, $v_i \leq_D v_k$ and $v_k \leq_D v_j$. Thus, there is a path in D from v_i to v_j traversing v_k. However, this path is distinct from $v_i \xrightarrow{e} v_j$, contradicting the definition of directed subgraphs. Therefore, e is a directed edge in D'. Similarly, we can demonstrate that every edge in D' also belongs to D with the same direction. Thus, $D = D'$. □

F.2 Proof of Theorem 3.2.1

Proof. The empty graph is its identity element, and the associativity of $\bullet$ comes from the associativity of the composition of paths. The non-commutativity is explained in Example 3.2.2. □

F.3 Proof of Theorem 3.2.5

Proof. Since directed subgraphs, together with the operation $\bullet$ generate the monoid $\mathsf{Mod}(G)$, we just need to show that every directed subgraph can be formed by its directed edges using the operation $\bullet$. We will prove this by induction based on the number of edges. Let D be a directed subgraph of G. There is nothing to prove if D has just one directed edge. Suppose the number of edges in D is m, and the statement is true for every directed subgraph with edges less than m; Our task is to show that the statement holds for D as well.

Let V_D be the set of nodes of D. Since $\leq_D$ is transitive, $(V_D, \leq_D)$ can be seen as a partially ordered set, implying the existence of maximal elements. A node is considered maximal if it is not the starting point of any path. Now, let v be a maximal node; we choose a directed edge $w \xrightarrow{e} v$ in D and remove it. The following three situations may occur:

1) producing one directed subgraph D': D and $D' \bigoplus e$ have the same directed edges. Since v is maximal, the paths of D that pass e have this directed edge as their terminal edge. Then

$$\mathsf{Paths}(D) = \mathsf{Paths}(D') \star e$$

This follows $D = D' \bullet e$. Based on the assumption, D' can be created by its edges. Then, the statement is true for D.

2) producing two components where one of them is an isolated node, and the other one is a directed subgraph D': in this case, we first remove the isolated node and then, similar to the first case, we conclude that the statement is true for D.

3) producing two directed subgraphs D' and D'' where $w \in D'$ and $v \in D''$: obviously D and $D' \bigoplus e \bigoplus D''$ have the same directed edges. With an argument similar to the first part, the maximality of v implies

$$\mathsf{Paths}(D) = \mathsf{Paths}(D') \star \{e\} \star \mathsf{Paths}(D'')$$

and then $D = D' \bullet e \bullet D''$. Now, by the assumption that D' and D'' can be created by their edges, the statement is true for D.

□

F.4 Proof of Theorem 3.3.1

Proof. Since the summation and multiplication of matrices are associative, the operation $\circ$ is associative. The zero matrix is the identity element of $\mathsf{Mat}_n(\mathbb{R})$ with respect to $\circ$. □

F.5 Proof of Theorem 3.3.3

To define a monoidal homomorphism between the monoids $(\mathsf{Mod}(G), \bullet)$ and $(\mathsf{Mom}(G), \circ)$ in such a way that it is an extension of the morphism Rep, we first prove the following theorem which gives a good explanation of the monoidal operation $\circ$.

Theorem F.5.1. *For $A_1, A_2, \cdots, A_k \in \mathsf{Mat}_n(\mathbb{R})$ with $k \in \mathbb{N}$ we have:*

$$A_1 \circ A_2 \circ \cdots \circ A_k = \sum_{i=1}^{k} A_i + \sum_{\sigma \in O(k,2)} A_{\sigma_1} A_{\sigma_2} + \cdots + \sum_{\sigma \in O(k,j)} A_{\sigma_1} \cdots A_{\sigma_j} + \cdots + A_1 A_2 \cdots A_k$$

where $O(k,i)$ is the set of all strictly monotonically increasing sequences of i numbers of $\{1, \cdots, k\}$

Proof. We prove the statement by induction on k. For $k = 2$, there is nothing to prove, which is clear from the definition. Let the statement be true for k; We will show it is true for $k+1$. The associativity of $\circ$ and the induction hypothesis imply:

$$A_1 \circ A_2 \circ \cdots \circ A_k \circ A_{k+1} = (A_1 \circ A_2 \circ \cdots \circ A_k) \circ A_{k+1} =$$

$$(A_1 \circ A_2 \circ \cdots \circ A_k) + A_{k+1} + (A_1 \circ A_2 \circ \cdots \circ A_k) A_{k+1} =$$

$$\sum_{i=1}^{k} A_i + \cdots + \sum_{\sigma \in O(k,j)} A_{\sigma_1} \cdots A_{\sigma_j} + \cdots + A_1 A_2 \cdots A_k +$$

$$A_{k+1} +$$

$$(\sum_{i=1}^{k} A_i + \cdots + \sum_{\sigma \in O(k,j)} A_{\sigma_1} \cdots A_{\sigma_j} + \cdots + A_1 \cdots A_k) A_{k+1}$$

$$= \sum_{i=1}^{k+1} A_i + (\sum_{i=1}^{k} A_i A_{k+1} + \sum_{\sigma \in O(k,2)} A_{\sigma_1} A_{\sigma_2}) + \cdots +$$

$$(\sum_{\sigma \in O(k,j-1)} A_{\sigma_1} \cdots A_{\sigma_{j-1}} A_{k+1} + \sum_{\sigma \in O(k,j)} A_{\sigma_1} \cdots A_{\sigma_j}) +$$

$$\cdots + A_1 \cdots A_k A_{k+1} =$$

$$\sum_{i=1}^{k+1} A_i + \sum_{\sigma \in O(k+1,2)} A_{\sigma_1} A_{\sigma_2} + \cdots + \sum_{\sigma \in O(k+1,j)} A_{\sigma_1} \cdots A_{\sigma_j} +$$

$$\cdots + A_1 A_2 \cdots A_k A_{k+1}$$

Therefore the statement is true for $k+1$. □

Now, we prove Theorem 3.3.3.

Proof. Considering that $S = \mathsf{Paths}(D_1) \star \cdots \star \mathsf{Paths}(D_k)$, let $p = p_0 p_1 \cdots p_m \in S$ be a path from v_i to v_j that is obtained by composition of subpaths $p_0 \in \mathsf{Paths}(D_{i_0}), \cdots, p_m \in \mathsf{Paths}(D_{i_m})$ and $1 \leq i_0 \lneqq \cdots \lneqq i_m \leq k$. The number of all such paths from v_i to v_j equals the ij entry of the matrix $(A_{i_0} \cdots A_{i_m})$ that is a summand of A as explained in Theorem F.5.1. So the number of all paths from v_i to v_j in S equals the ij entry of A. Therefore, the definition of Tr just depends on S and is independent of the choice of D_is. Then Tr is well-defined. Based on the definition, Tr is a monoidal homomorphism.

Suppose $B \in \mathsf{Mom}(G)$, then there are some matrix representations $B_1, \cdots, B_l$ in $\mathsf{MatRep}(G)$ such that $B = B_1 \circ \cdots \circ B_l$. Since Rep is an isomorphism, there exist some directed subgraphs $C_1, \cdots, C_l$ such that $\mathsf{Rep}(C_i) = B_i$. Now, by choosing $C = C_1 \bullet \cdots \bullet C_l$, we obtain $\mathsf{Tr}(C) = B$, establishing that Tr is surjective. □

F.6 Proof of Proposition 4.1.1

Proof. As we explained, f changes the order of rows and columns. Thus, it preserves element-wise and matrix multiplications. Since f is also linear, we have

$$\begin{aligned} f(A \circ B) &= f(A + B + AB) \\ &= f(A) + f(B) + f(AB) \\ &= f(A) + f(B) + f(A)f(B) \\ &= f(A) \circ f(B) \end{aligned}$$

and then f preserves the operation $\circ$ and this property establishes f as a monoidal isomorphism. □

F.7 Proof of Theorem 4.1.2

Proof. Since f is a change in the order, it induces bijections $\mathsf{DirSub}(f)$ and $\mathsf{MatRep}(f)$ such that Diagram 6 commutes.

$$\begin{tikzcd} \mathsf{DirSub}(G) \arrow[r, "\mathsf{Rep}"] \arrow[d, "\mathsf{DirSub}(f)"'] & \mathsf{MatRep}(G) \arrow[d, "\mathsf{MatRep}(f)"] \\ \mathsf{DirSub}(H) \arrow[r, "\mathsf{Rep}"'] & \mathsf{MatRep}(H) \end{tikzcd} \tag{6}$$

Also, f induces monoidal isomorphism $\mathsf{SMult}(f) : \mathsf{SMult}(G) \to \mathsf{SMult}(H)$ that sends $(M, S) \mapsto (f(M), f(S))$. According to the commutativity of the squares in Diagram 7, isomorphisms $\mathsf{Mod}(f) : \mathsf{Mod}(G) \to \mathsf{Mod}(H)$ and $\mathsf{Mom}(f) : \mathsf{Mom}(G) \to \mathsf{Mom}(H)$ can be obtained by restricting $\mathsf{SMult}(f)$ to $\mathsf{Mod}(G)$ and $\mathsf{CO}(f)$ to $\mathsf{Mom}(G)$.

$$\begin{tikzcd} \mathsf{DirSub}(G) \arrow[r, "\mathsf{DirSub}(f)"] \arrow[d] & \mathsf{DirSub}(H) \arrow[d] & \mathsf{MatRep}(G) \arrow[r, "\mathsf{MatRep}(f)"] \arrow[d] & \mathsf{MatRep}(H) \arrow[d] \\ \mathsf{SMult}(G) \arrow[r, "\mathsf{SMult}(f)"'] & \mathsf{SMult}(H) & \mathsf{Mat}_{\lvert V_G \rvert}(\mathbb{R}) \arrow[r, "\mathsf{CO}(f)"'] & \mathsf{Mat}_{\lvert V_H \rvert}(\mathbb{R}) \end{tikzcd} \tag{7}$$

The commutativity of the right square in Diagram 1 directly follows from the definition of $\mathsf{Mom}(f)$. As illustrated in Diagram 6, the left square in Diagram 1 is shown to be commutative for the generators of monoids, establishing the commutativity of this square. □

F.8 Proof of Theorem 4.1.3

Proof. We begin by demonstrating that f establishes a one-to-one correspondence between the edges of G and H. It is evident that a matrix with a single non-zero entry in either $\mathsf{Mom}(G)$ or $\mathsf{Mom}(H)$ corresponds to a matrix transformation of an element in $\mathsf{Mod}(G)$ or $\mathsf{Mod}(H)$, respectively, each representing a single directed edge.

For an edge $v_i \text{———} v_j$ in G, let e be the directed edge $v_i \to v_j \in \mathsf{Mod}(G)$; then $A = \mathsf{Tr}_G(e)$ has one non-zero entry, and since f is a linear isomorphism, $f(A)$ has one non-zero entry, and, based on the assumption, it belongs to $\mathsf{Mom}(H)$. So $f(A)$ is a matrix transformation of a directed edge $c : u_k \to u_l$ in $\mathsf{Mod}(H)$. Similarly, let $B \in \mathsf{Mom}(G)$ be the matrix transformation of $e' : v_j \to v_i$ and then $f(B) \in \mathsf{Mom}(H)$ is a matrix transformation of some directed edge $c' : u_{l'} \to u_{k'}$ in $\mathsf{Mod}(H)$. Since e can be followed by e', $e \bullet e'$ has three paths. This implies $\mathsf{Tr}_G(e \bullet e')$ has three non-zero entries. On the other hand, $\mathsf{Tr}_G(e \bullet e') = \mathsf{Tr}_G(e) \circ \mathsf{Tr}_G(e') = A \circ B = A + B + AB$; then $AB \neq 0$ and consequently $f(A)f(B) = f(AB) \neq 0$. The equation

$$\begin{aligned} \mathsf{Tr}_H(c \bullet c') &= \mathsf{Tr}_H(c) \circ \mathsf{Tr}_H(c') \\ &= f(A) \circ f(B) \\ &= f(A) + f(B) + f(A)f(B) \end{aligned}$$

says that the matrix transformation corresponding to $c \bullet c'$ has three non-zero entries and so $c \bullet c'$ contains three paths. Then c must be followed by c' and this yields $u_l = u_{l'}$. Similarly, $u_k = u_{k'}$ can be shown. Therefore, f gives a one-to-one mapping between the edges of G and H.

To prove the correspondence between the nodes of two graphs, let v_x be a node in G, connected to v_i in which $j \neq x$ and C and $f(C)$ be the matrix transformations of $a : v_i \to v_x \in \mathsf{Mod}(G)$ and $b : u_y \to u_z \in \mathsf{Mod}(H)$, respectively. Since e' is followed by a in $\mathsf{Mod}(G)$, with the same reasoning as above, c' must be followed by b in $\mathsf{Mod}(H)$ and this means $u_k = u_y$. So f also gives a one-to-one mapping between nodes of graphs compatible with edges. Then, G and H are isomorphic. □

F.9 Proof of Theorem 4.2.2

The role of neighborhoods in MPNN is like a sink such that messages move to the center of the sink. For a node v_k with neighborhood N_k containing $v_{k_1}, v_{k_2}, \cdots, v_{k_m}$, we depict this sink in Figure 5 by denoting directed edge from v_{k_i} to v_k by $e_i : v_{k_i} \to v_k$. This sink can be considered as a directed subgraph. As an

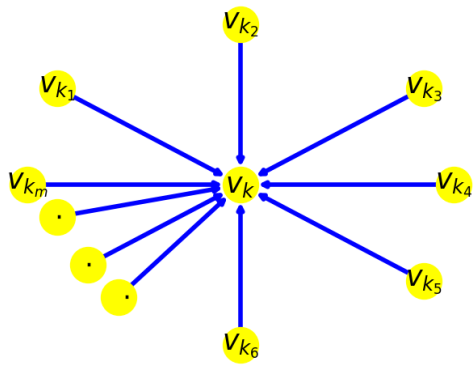

Figure 5: Visualizing a neighborhood by representing it as a directed subgraph

element of $\mathsf{Mod}(G)$, it can be represented as follows:

$$S_k = e_1 \bullet e_2 \bullet \cdots \bullet e_m$$

Since the directed edges e_i and e_j appearing in S_k are not composable, we observe $e_i \bullet e_j = e_j \bullet e_i$, rendering the order in S_k unimportant. The cover obtained by S_ks is exactly the cover of the neighborhoods. Let $T_k = \mathsf{Tr}(S_k)$ and $A_i = \mathsf{Tr}(e_i)$. Thus A_i has 1 in the entry $k_i k$ and 0 for all other entries. The matrix transformation of $e_i \bullet e_j$ has just two non-zero entries and $\mathsf{Tr}(e_i \bullet e_j) = A_i + A_j + A_i A_j$. Then $A_i A_j = 0$ for $1 \leq i \leq m$ and $1 \leq j \leq m$. Theorem F.5.1 implies

$$\begin{aligned} T_k = \mathsf{Tr}(S_k) &= A_1 \circ A_2 \circ \cdots \circ A_m \\ &= A_1 + A_2 + \cdots + A_m \end{aligned}$$

As a result, the column k of T_k aligns with the column k of the adjacency matrix of graph G, while the remaining columns are filled with zeros. Transforming the cover $\{S_k\}$ yields a collection of $|V|$ matrices, each containing a single column from the adjacency matrix. In the GkGNN framework, summation is an allowed operation, enabling the construction of the adjacency matrix by performing the summation on this matrix collection. Hence, neighborhoods can function as a cover within the framework of GkGNN, with the adjacency matrix serving as an interpretation of this cover.

F.10 Proof of Theorem 5.1.1

Proof. Since the definition of sets $M_i(v)$s is based on the neighborhoods, for a graph isomorphism $f : G \to H$, $f(M_i(v)) = M_i(f(v))$. This follows $\mathsf{Mod}(f)(D_i(v)) = D_i(f(v))$. Since $\mathsf{Mod}(f)$ is a monoidal homomorphism, we get:

$$\begin{aligned}
\mathsf{Mod}(f)(\mathsf{Sieve}(v,k)) &= \mathsf{Mod}(f)(D_k(v) \bullet \cdots \bullet D_0(v)) \\
&= \mathsf{Mod}(f)(D_k(v)) \bullet \cdots \bullet \mathsf{Mod}(f)(D_0(v)) \\
&= D_k(f(v)) \bullet \cdots \bullet D_0(f(v)) \\
&= \mathsf{Sieve}(f(v),k)
\end{aligned}$$

Based on Theorem 4.1.2, $\mathsf{Mom}(f)(\mathsf{Image}(v,k)) = \mathsf{Image}(f(v),k)$. □

F.11 Proof of Theorem 5.2.1

Proof. Since the cover of sieves is invariant and $\mathsf{CO}(f)$ preserves the rest of the computations in the algorithm, SNN is invariant. □

F.12 Proof of Theorem E.1.2

Proof. Let $\mathsf{Adj}(v)$ denote the matrix representation of the neighborhood of a node $v \in G$. As demonstrated, this matrix contains exactly one non-zero column. The mapping f is a Change-of-Order mapping, which transforms $\mathsf{Adj}(v)$ into a matrix with a single non-zero column, where all non-zero entries are equal to 1.

An element of $\mathsf{Adj}(H)$ that is not a matrix transformation of any element in the cover of the neighborhood will have two or more non-zero columns. Consequently, for $f(\mathsf{Adj}(v)) \in \mathsf{Adj}(H)$, there exists a node $u \in H$ such that $f(\mathsf{Adj}(v)) = \mathsf{Adj}(u)$.

This establishes a one-to-one correspondence between V_G and V_H, as f is an isomorphism. Now, let v_i —— v_j represent an edge in G, with $f(\mathsf{Adj}(v_i)) = \mathsf{Adj}(u_k)$ and $f(\mathsf{Adj}(v_j)) = \mathsf{Adj}(u_l)$. The entry ii in the matrix $\mathsf{Adj}(v_j) \circ \mathsf{Adj}(v_i)$ equals 1.

Since f is a Change-of-Order mapping, the matrix $f(\mathsf{Adj}(v_j) \circ \mathsf{Adj}(v_i)) = \mathsf{Adj}(u_l) \circ \mathsf{Adj}(u_k)$ has a diagonal entry equal to 1. In this matrix, the only diagonal entry that can be non-zero is the entry kk. Similarly, the entry ll in $\mathsf{Adj}(u_k) \circ \mathsf{Adj}(u_l)$ equals 1. This implies that there is an edge between u_k and u_l.

Thus, we establish a one-to-one correspondence between the edges of G and H that is consistent with the mapping of their nodes. This proves that f defines a graph isomorphism between G and H. □