RELAXING PERMUTATION EQUIVARIANCE IN GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph Neural Networks (GNNs) have been established as powerful methods for learning functions on graph-structured data, in which nodes carry no canonical ordering. Permutation equivariance to node index relabelling is therefore a defining inductive bias of GNNs. We argue that the representation of nodes in sets is limiting. We propose an alternative approach that represents and learns on observed graphs, in which nodes are augmented with one or several partial orders. Leveraging the equivalence between partially ordered sets and Directed Acyclic Graphs (DAGs), we introduce a new data structure called Partially Ordered Graphs (POGs). Building on theoretical work defining convolution operators on DAGs, we construct the Partially Ordered GNN (POGNN) architecture which combines trainable convolutions on both the node partially ordered sets and the underlying observed graph. We also prove that our POGNNs relax the permutation equivariance assumption of traditional GNNs in a tunable manner by being equivariant to the automorphism of the poset. The choice of partial order allows us to cover the full spectrum from node sets to node sequences. In practice, we demonstrate that our model improves the performance of traditional GNN approaches on several graph-level biochemical benchmark datasets.

1 INTRODUCTION

Graph Neural Networks (GNNs) have emerged as a powerful approach to learning functions on graph-structured data, with successes in large-scale industrial contexts ranging from Google Maps, LinkedIn, to Spotify (Derrow-Pinion et al., 2021; Borisyuk et al., 2024; De Nadai et al., 2024). To handle the arbitrary node indexing of graphs, GNNs are typically designed to be permutation equivariant. The standard way GNNs achieve permutation equivariance is by employing neighborhood aggregation schemes (Kipf & Welling, 2017; Gilmer et al., 2017) in which the aggregation is locally permutation invariant and, hence, globally acts in a permutation equivariant way. While this property provides a robust geometric prior and facilitates generalisation (Wu et al., 2020), it can limit empirical performance and faithfulness to the underlying data in practice.

Much of the success of deep learning stems from encoding the appropriate inductive bias in the network architecture (e.g., translation invariance in image recognition (LeCun et al., 2002), long-term memory in natural language processing (Hochreiter & Schmidhuber, 1997)). However, as argued in a recent survey (Veličković, 2023), the permutation equivariance assumption might not be the right inductive bias for every observed graph. We argue that the underlying mechanism generating graphs often exhibits natural partial orders that can be leveraged to encode richer structural information.

We illustrate our idea using the example of molecule data. Modeling a molecule as a graph, i.e., as a set of nodes with connectivity, treats node indices as interchangeable, so a GNN is equivariant to every permutation of the atoms' indices. Yet, in practice, these labels represent atoms that can be distinguished by their atomic number (e.g., 1 for hydrogen, 6 for carbon, 8 for oxygen) and should not be interchangeable. A visual explanation is given in Figure 1. Standard GNNs account for this through node features in which the atomic number can be provided. Once atomic numbers are given as input, the only permutations that leave the output unchanged are those exchanging atoms of the same chemical element. The symmetry is thus broken at the level of the features. Instead, we propose to encode this relaxation structurally. Atomic numbers naturally induce a partial order on atoms, and

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

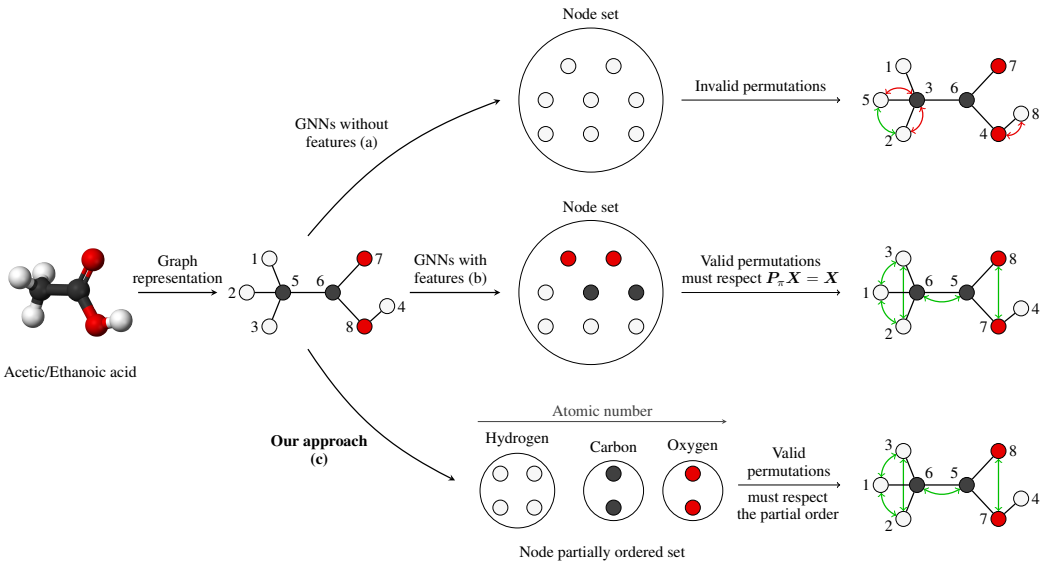

Figure 1: Symmetries of acetic acid (4 hydrogen, 2 carbon and 2 oxygen atoms) under three views of the same molecular graph, shown with an arbitrary initial node indexing. (a) Without node features, the graph is a set of interchangeable nodes. A GNN therefore treats every relabeling as a symmetry, including chemically invalid ones that exchange atoms of different elements. (b) When atomic numbers are given as node features, only permutations satisfying $P_\pi X = X$, i.e., those exchanging atoms of the same element, leave the output unchanged. The symmetry is broken at the level of the input features. (c) Our approach orders atoms by atomic number (H $\prec$ C $\prec$ O), turning the node set into a node poset. A POGNN is equivariant only to the automorphisms of this poset, so the symmetry is instead structurally relaxed in the architecture itself.

a model that respects this order is equivariant to the permutations that preserve it. Further motivation is presented in Appendix A.

To this end, we introduce *Partially Ordered Graphs*, a data structure that augments an observed graph with an auxiliary node partially ordered set (poset) represented as a Directed Acyclic Graph (DAG). To learn node representations on this data structure, we propose *Partially Ordered Graph Neural Networks (POGNNs)*. They leverage the node partial order to relax the permutation equivariance assumption in GNNs by being equivariant to permutations that preserve the partial order (automorphisms of the poset), hence combining graph convolution with DAG convolution. Our approach opens up a spectrum of representations: from graphs as unordered node sets (canonically processed by GNNs) to graphs as totally ordered node sequences (best processed by Transformers or Recurrent Neural Networks), targeting the intermediate regime of posets.

Contributions. The main contributions of our work are as follows:

- We define Partially Ordered Graphs (POGs), a new data structure that augments a standard graph with an auxiliary DAG derived from a node partial order, adding information that is not captured by purely relational representations.
- We introduce the Partially Ordered Graph Neural Network (POGNN), a theoretically motivated architecture that relaxes strict permutation equivariance by being equivariant to the permutations in the automorphism of the node poset.
- We empirically validate our approach through experiments on graph-level biochemical tasks, showing that it improves the performance of traditional GNNs and Graph Transformers.

2 Related Work

We propose to discuss two streams of work related to our approach: methods that relax the permutation equivariance in GNNs, and methods that exploit partial-order or equivalently DAG structure to bias learning towards directed dependencies.

Relaxed Equivariance Architectures in Graph Neural Networks. Message-passing GNNs (Gilmer et al., 2017; Kipf & Welling, 2017; Veličković et al., 2018; Xu et al., 2019; Bresson & Laurent, 2017) have been successfully applied to a range of tasks on graphs. Typical examples include molecular property prediction (Hu et al., 2020a), citation graph classification (Sen et al., 2008; Yang et al., 2016), and social network modeling (Hamilton et al., 2017; Rossi et al., 2020). Even though the permutation equivariance bias in GNNs provides a robust geometric prior and facilitates generalisation (Wu et al., 2020), it can limit faithfulness to the underlying data and, hence, several approaches to reducing this inductive bias have been explored. Existing relaxations differ in where the reduced symmetry comes from and in what sense permutation equivariance is weakened. One line of work aims to design models that learn an enlarged function class: Natural Graph Networks (de Haan et al., 2020) replace permutation equivariance by naturality, requiring only that isomorphic graphs be processed compatibly. Huang et al. (2023) instead select a symmetry group between the graph automorphisms and the symmetric group, S_n, obtained from automorphisms of a coarsening. A second line of work modifies the model input and thereby targets Curie's principle, that an equivariant map cannot output less symmetry than it receives (Kaba & Ravanbakhsh, 2023). They weaken the constraint to relaxed equivariance, satisfied up to the input's stabiliser, while Lawrence et al. (2025) recover symmetry breaking probabilistically, sampling a canonicalising group element as a positional encoding. Relatedly, Dym et al. (2024) show unweighted frames cannot canonicalise S_n continuously, motivating weighted frames.

Learning on Directed Acyclic Graphs. A different stream of research develops signal processing and learning techniques on partially ordered sets, equivalently modeled as DAGs. Seifert et al. (2023) introduce a Fourier transform for DAG signals by constructing a shift operator from the transitive closure, giving a principled spectral framework for convolution on DAGs that connects to Möbius inversion in poset theory. Building on it, Directed Acyclic Graph Convolutional Networks (DCN) (Rey et al., 2026) stack causal graph filters that respect the partial order, yielding representations biased toward causal ancestors, together with a parallel variant (PDCN) that improves efficiency. Differently, Directed Acyclic Graph Neural Networks (DAGNN) (Thost & Chen, 2021) enforce the partial order in message passing itself, letting each node aggregate only from its predecessors, which generalises recurrent DAG and tree models and outperforms generic GNNs and Bayesian networks on program graphs. Moving beyond message passing, DAGformer (Luo et al., 2023) injects node order into attention, restricting each node to attend to its ancestors and descendants under the reachability relation and adding a positional encoding based on topological levels; being only a constraint on which pairs interact, it applies to any transformer-based graph model. To the best of our knowledge, this literature has not considered a data structure in which an underlying graph is augmented with a node poset.

Our work combines the two approaches: enriching the input with a node partial order, modeled as a DAG, and designing a model equivariant to the poset automorphisms rather than the symmetric group as in GNNs, thereby enlarging the learnable function class while retaining exact equivariance to a group derived from task knowledge rather than from the graph formalism.

3 Preliminaries

This section sets up the notation used throughout the paper. We first review permutation symmetries and message passing on graphs, and then summarise the framework of signal processing on DAGs, which forms the basis of our DAG convolution layer.

Permutation Invariance and Equivariance in Graphs. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph, where $\mathcal{V}$ represents the set of vertices and $\mathcal{E}$ represents the set of edges. The connectivity of the graph is encoded in the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, where $n = |\mathcal{V}|$, and the neighborhood of a node u is defined as $N(u) = \{v : (u, v) \in \mathcal{E}\}$. Additionally, in some cases, nodes may have associated feature vectors $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{in}}}$ where d_{in} is the dimensionality of the input features.

Any relabeling of the nodes corresponds to a permutation $\pi \in S_n$, with S_n denoting the symmetric group on n elements, which can be represented by a permutation matrix $\boldsymbol{P}_\pi \in \{0,1\}^{n\times n}$. Any given graph $(\boldsymbol{X}, \boldsymbol{A})$ can hence be represented with another feature matrix and adjacency matrix related to the original matrices by permutation: $(\boldsymbol{P}_\pi \boldsymbol{X}, \boldsymbol{P}_\pi \boldsymbol{A} \boldsymbol{P}_\pi^\top)$.

Definition 1 (Permutation Invariance and Equivariance on Graphs)**.** A function operating on a graph $f:\ (\boldsymbol{X}, \boldsymbol{A}) \to \boldsymbol{Z}$, where $\boldsymbol{Z} \in \mathbb{R}^{d_{\text{out}}}$ denotes transformed node embeddings, is called permutation invariant if for all $\pi \in S_n$

$$f\left(\boldsymbol{P}_\pi \boldsymbol{X}, \boldsymbol{P}_\pi \boldsymbol{A} \boldsymbol{P}_\pi^\top\right) = f\left(\boldsymbol{X}, \boldsymbol{A}\right).$$

A function operating on a graph $f:\ (\boldsymbol{X}, \boldsymbol{A}) \to\ \boldsymbol{Z}$, where $\boldsymbol{Z} \in \mathbb{R}^{n\times d_{\text{out}}}$ denotes transformed node embeddings, is called permutation equivariant if for all $\pi \in S_n$

$$f\left(\boldsymbol{P}_\pi \boldsymbol{X}, \boldsymbol{P}_\pi \boldsymbol{A} \boldsymbol{P}_\pi^\top\right) = \boldsymbol{P}_\pi f\left(\boldsymbol{X}, \boldsymbol{A}\right).$$

Equivariance ensures that the output embeddings respect the symmetries of the input: relabeling nodes before processing the graph is equivalent to relabeling them after processing. This property is crucial in machine learning on graphs because node identifiers are arbitrary, and meaningful graph structure must be equivariant to permutations of node ordering.

Message Passing in Graph Neural Networks. A central inductive bias of most GNNs is the assumption of permutation equivariance. The standard way GNNs achieve permutation equivariance is by employing neighborhood aggregation schemes (Kipf & Welling, 2017; Gilmer et al., 2017). At the ℓ-th layer, the embedding of node u, $\boldsymbol{h}_u^{(\ell)}$, is updated as

$$\boldsymbol{h}_u^{(\ell+1)} = \phi\left(\boldsymbol{h}_u^{(\ell)}, \bigoplus_{v\in N(u)} \psi\left(\boldsymbol{h}_u^{(\ell)}, \boldsymbol{h}_v^{(\ell)}, \boldsymbol{e}_{uv}\right)\right),$$

where $\boldsymbol{e}_{uv}$ are edge features, $\oplus$ is a permutation-invariant multiset function or aggregation (typically sum, mean, or max), and ϕ, ψ are learnable functions (often implemented as MLPs). Since $\oplus$ is symmetric with respect to the ordering of neighbors, the update rule satisfies permutation equivariance by construction.

Signal Processing on Partially Ordered Sets. Our approach builds upon the equivalence between DAGs and posets. As explained in Seifert et al. (2023), every DAG $\mathcal{D} = (\mathcal{V}, \mathcal{E})$ induces a unique partial order on $\mathcal{V}$, defined as $u \prec v$ whenever there exists a directed path from u to v. Conversely, given a poset $\mathcal{P}$, there are in general many DAGs that $\mathcal{P}$ can induce, but two DAG constructions are canonical and unique. The first is the reachability graph, which is transitively closed (a property usually denoted by an overline), meaning that whenever there is a path from y to x, there is also an edge (y, x) in the DAG. It is defined as $\overline{\mathcal{D}} = (\mathcal{P}, \overline{\mathcal{E}})$ with $\overline{\mathcal{E}} = \{(u, v) : u \prec v\}$. The second is the cover graph, which is built from the fact that u is covered by v, denoted by $u \prec\cdot\ v$, if $u \prec v$ and there is no u' such that $u \prec u' \prec v$. The cover graph is transitively reduced, meaning that it is the DAG with the fewest number of edges inducing $\mathcal{P}$. It is defined as $\mathcal{D} = (\mathcal{P}, \mathcal{E})$ with $\mathcal{E} = \{(u, v) : u \prec\cdot\ v\}$. Hence, DAGs and posets are interchangeable representations of partial orders, with different choices of edge drawing rules leading to either redundant but complete (transitively closed) or minimal (transitively reduced) encodings.

Transitive Closure Operation. In a DAG, transitivity propagates influence beyond direct edges. That is to say, if u is a parent of v and v is a parent of u', then u will also influence u'. The transitive closure operation formalises this transitivity property by assigning a weight $w_{u,u'}$ that aggregates the contributions of all directed paths from u to u' with two operations: $\odot$ that combines weights along a path and $\oplus$ that combines parallel paths (see Figure 5 in Appendix B).

4 Partially Ordered Graphs

Many graphs admit natural order relations among their nodes that are not fully captured by the node set formalism. In this section, we formalise how such partial orderings can be integrated into graph representations. We introduce a new data structure, the *Partially Ordered Graph (POG)*, which augments a graph with a partial order on its nodes encoded as a DAG. This structure will serve as the foundation for defining convolutional operators on DAGs in the next section.

Node Partially Ordered Sets. The constraint of permutation equivariance in GNNs, defined in Definition 1, stems from the fact that graphs are node sets with connectivity. However, graphs often exhibit natural partial orderings that can be leveraged to encode additional structural information. Examples include orderings induced by temporal information, structure- or feature-based rankings and underlying mechanisms such as peptide sequences in proteins. Since posets are an intermediate structure between sets and sequences, node posets arise as a natural way to relax the permutation equivariance as modeling a graph with a node sequence would completely break equivariance.

Definition 2 (Partially Ordered Graph)**.** Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with node set $\mathcal{V}$ and edge set $\mathcal{E}$ and $\mathcal{P} = (\mathcal{V}, \preceq)$ a partial order on $\mathcal{V}$. A *Partially Ordered Graph (POG)* is a tuple

$$\mathcal{G}_{\preceq} = (\mathcal{V}, \mathcal{E}, \mathcal{E}_{\mathcal{D}}),$$

where $\mathcal{E}_{\mathcal{D}} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set of the cover DAG $\mathcal{D} = (\mathcal{V}, \mathcal{E}_{\mathcal{D}})$ that encodes $\mathcal{P}$, i.e., $u \prec \cdot\, v \iff (u, v) \in \mathcal{E}_{\mathcal{D}}$.

In partially ordered graphs, the $\mathcal{G}$ captures the relational structure of the data, while the DAG $\mathcal{D}$ encodes a partial order on the same node set, thereby enriching the representation. This data structure provides a natural way to relax strict permutation equivariance. An example of how a partially ordered graph is inferred from a graph and a partial order on nodes is given in Appendix C.

To enable Fourier analysis and convolution on DAGs, we require the adjacency matrix of $\mathcal{D}$ to be triangular and, therefore, the node indexing must respect the partial order. Thus, we assign node indices according to a topological ordering (linear extension) of $\mathcal{P}$. This is achieved by applying Kahn's algorithm (Kahn, 1962) on $\mathcal{E}_{\mathcal{D}}$, and then permuting the feature and adjacency matrices of the original graph accordingly.

Transitive Closure Choice. The Fourier analysis on DAGs is highly dependent on the choice of transitive closure (Seifert et al., 2023). In this work, we use the multiplicative closure $\boldsymbol{A}_c = (\boldsymbol{I} - \boldsymbol{A}_r)^{-1}$ where $\boldsymbol{A}_r$ is the adjacency matrix of the transitively reduced version of the DAG. This choice has two motivations. First, it admits a natural information-propagation interpretation since $\boldsymbol{A}_c[u, u']$ aggregates contributions of all directed paths from u to u', multiplicatively along each path and additively across parallel paths, in contrast to min/max aggregations which destroy information to a single representative path. Second, the set of causal graph shift operators derived from this weight transitive closure has been shown to be unique (Rey et al., 2026, Proposition 2), i.e., non-isomorphic DAGs yield different causal graph shift operator matrices. This result is useful in the sense that different orderings applied to the nodes must yield different graph shift operators to encode different information. Alternative aggregation choices (or semirings) yield transitive closures with different interpretations, and we summarise these in Appendix B.

5 Partially Ordered Graph Neural Networks

In Section 4, we introduced partially ordered graphs as data structures that enrich standard graphs with additional partial ordering information. We now describe how to design neural architectures that can effectively learn from such structures. Our approach builds on recent advances in Fourier analysis on DAGs and in graph signal processing. It extends them to more scalable formulations, then uses them to learn on Partially Ordered Graphs.

5.1 Learning on Directed Acyclic Graphs

Prior work by Rey et al. (2026), building on Seifert et al. (2023), formulates the output $\boldsymbol{H}^{(\ell+1)} \in \mathbb{R}^{n \times d_{out}^{(\ell)}}$ of the ℓ-th convolutional layer on DAGs in a stack of multiple DAG perceptrons for a bank of learnable DAG filters as:

$$\boldsymbol{H}^{(\ell+1)} = \sigma\left(\sum_{u \in \mathcal{V}} \boldsymbol{S}_u \boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_u^{(\ell)}\right), \tag{1}$$

where $\boldsymbol{S}_u$ denotes the causal graph shift operator associated with node u, $\boldsymbol{\Theta}_u^{(\ell)} \in \mathbb{R}^{d_{\text{in}}^{(\ell)} \times d_{\text{out}}^{(\ell)}}$ contains the learnable filter coefficients, $d_{\text{in}}^{(\ell)}$ and $d_{\text{out}}^{(\ell)}$ respectively denote the number of input and output features, and σ is a non-linear activation function (such as the ReLU).

Our formulation. Since Equation (1) requires storing all the matrices $\boldsymbol{S}_u$ corresponding to graph shift operators for each node in memory, it is memory-intensive when used on medium to large DAGs or POGs. To tackle this memory issue, we propose a new matrix formulation for the convolution of a signal $\boldsymbol{H}^{(\ell)} \in \mathbb{R}^{n \times d_{in}^{(\ell)}}$ on a DAG by a filter $\theta = (\theta_1, \ldots, \theta_n)$ (see Appendix D for proof)

$$\boldsymbol{X}_{\text{conv}} = \boldsymbol{A}_c \boldsymbol{\Lambda} (\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{X},$$

where $\boldsymbol{A}_c$ denotes the adjacency matrix of the closed DAG, $\boldsymbol{A}_r$ denotes the adjacency matrix of the reduced DAG and $\boldsymbol{\Lambda}$ is the diagonal matrix corresponding to the filter coefficients such that $\boldsymbol{\Lambda}_{i,i} = \theta'_i$.

However, learning one filter coefficient per node can become computationally expensive for large graphs and, more importantly, does not allow generalisation across graphs with different numbers of nodes. That is why we introduce a linear layer $\boldsymbol{W}_{\text{filter}}^{(\ell)} \in \mathbb{R}^{d_{\text{in}}^{(\ell)} \times 1}$ such that $\boldsymbol{X}\boldsymbol{W}_{\text{filter}}^{(\ell)} = (\theta_u)_{u \in \mathcal{V}} \in \mathbb{R}^n$. The convolution of a signal $\boldsymbol{H}^{(\ell)}$ by a learnable filter, then becomes

$$\boldsymbol{H}^{(\ell+1)} = \sigma\left(\boldsymbol{A}_c \operatorname{diag}\left(\boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\text{filter}}^{(\ell)}\right) (\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)}\right), \tag{2}$$

where $\boldsymbol{A}_c$ denotes the weight transitive closure matrix of the DAG, $\boldsymbol{W}_{\text{filter}}^{(\ell)} \in \mathbb{R}^{d_{\text{in}}^{(\ell)}}$ is a linear layer that computes the filter coefficients and $\boldsymbol{W}_{\mathcal{D}}^{(\ell)} \in \mathbb{R}^{d_{\text{in}}^{(\ell)} \times d_{\text{out}}^{(\ell)}}$ is a projection layer onto the output dimension of layer ℓ.

Equation (2) can be generalised to a bank of learnable filters as follows

$$\boldsymbol{H}^{(\ell+1)} = \sigma\left(\boldsymbol{A}_c \left[\left(\boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)}\right) \odot \left((\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)}\right)\right]\right), \tag{3}$$

where $\boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)} \in \mathbb{R}^{d_{\text{in}}^{(\ell)} \times d_{\text{out}}^{(\ell)}}$ is the linear layer computing for each output channel j, the per-node filter coefficients (column j of $\boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)} \in \mathbb{R}^{n \times d_{\text{out}}^{(\ell)}}$), and $\odot$ is the element-wise product between matrices.

Our modification to this convolution preserve the result of (Rey et al., 2026, Theorem 1) that such a layer is equivariant to permutations in the automorphisms of the poset, $\operatorname{Aut}(\mathcal{P})$, . This result will be essential in understanding our model and proving that it relaxes the permutation equivariance assumption of traditional GNNs.

5.2 Learning on Partially Ordered Graphs

We now discuss how learning convolutional filters on DAGs, combined with learning on graphs, helps us define a model that learns on partially ordered graphs. We furthermore demonstrate how this model relaxes the permutation equivariance assumption that GNNs satisfy.

The architecture we propose, *Partially Ordered Graph Neural Networks (POGNNs)*, takes as input a partially ordered graph, as defined in Section 4, and outputs its updated features. It combines DAG convolution filters, as in Equation (3), with traditional GNN layers and then applies a non-linear function σ (usually ReLU activation)

$$\boldsymbol{H}^{(\ell+1)} = \sigma\left(\text{GNN}^{(\ell)}(\boldsymbol{H}^{(\ell)}, \boldsymbol{A}) \oplus \left(\boldsymbol{A}_c \left[\left(\boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)}\right) \odot \left((\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)}\right)\right]\right)\right), \tag{4}$$

where $\oplus$ denotes an aggregator between the graph convolution and the DAG convolution, such as the sum or concatenation operation.

For instance, using the graph convolution of Kipf & Welling (2017) yields

$$\boldsymbol{H}^{(\ell+1)} = \sigma\Big(\underbrace{\widetilde{\boldsymbol{D}}^{-1/2} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-1/2} \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{G}}^{(\ell)}}_{\text{graph convolution}} \oplus \underbrace{\boldsymbol{A}_c \left[\left(\boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)}\right) \odot \left((\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)}\right)\right]}_{\text{DAG convolution}}\Big), \tag{5}$$

where $\widetilde{\boldsymbol{A}}$ and $\widetilde{\boldsymbol{D}}$ denote the normalised adjacency matrix and degree matrix of the input graph, respectively, and $\boldsymbol{W}_{\mathcal{G}}^{(\ell)}$ denotes the projection layer onto the output dimension of layer ℓ for the graph convolution layer.

The initial input, $\boldsymbol{H}^{(0)} = \boldsymbol{X}$, to the first layer, represents the graph signal $\boldsymbol{X}$ that we aim to process using the POGNN. The final layer's output, $\boldsymbol{H}^{(L)}$, is a graph signal containing node representations suitable for various downstream tasks (see Figure 2).

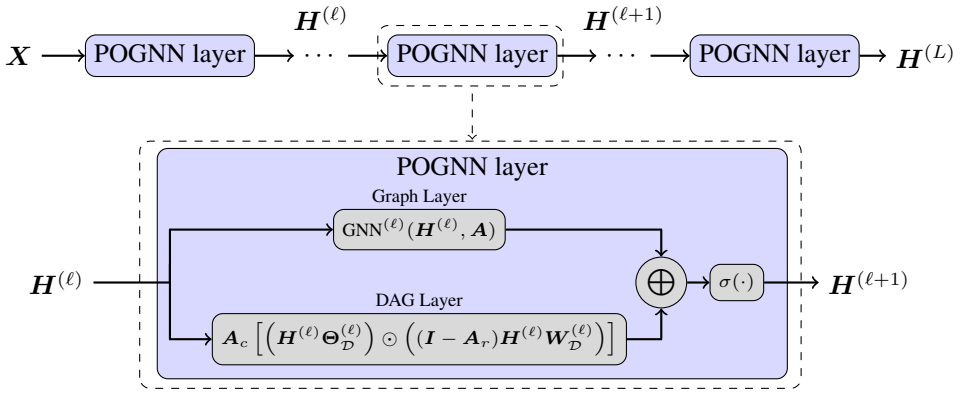

Figure 2: POGNN architecture where the GNN layer is agnostic and any graph learning model can be used.

Multi-Headed Partial Ordering. To capture diverse structures within the same graph, we extend our architecture by computing multiple DAG convolutions in parallel that represent different node partially ordered sets. Given M distinct partial orders, we instantiate a separate transitive closure matrix $\boldsymbol{A}_{c,m}$ for each head $m \in \{1, \ldots, M\}$. The multi-headed DAG convolution is defined by concatenating the representations:

$$\boldsymbol{H}_{\text{DAG}}^{(\ell+1)} = \left[\boldsymbol{h}_1^{(\ell+1)} \| \boldsymbol{h}_2^{(\ell+1)} \| \cdots \| \boldsymbol{h}_M^{(\ell+1)} \right], \tag{6}$$

where $\|$ denotes concatenation and $\boldsymbol{h}_m^{(\ell+1)} = \boldsymbol{A}_{c,m} \left[\left(\boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D},m}^{(\ell)} \right) \odot \left((\boldsymbol{I} - \boldsymbol{A}_{r,m}) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)} \right) \right]$.

Since each partial order $\mathcal{P}_m$ generally induces different node indices (linear extension), each head is computed in its own topological ordering permuted by π_m from a common original node ordering (the linear extension of an arbitrary reference poset). Its output is then mapped back to a common reference indexing, via $P_{\pi_m}^{\top}$ before concatenation.

5.3 Theoretical Analysis

The POGNN architecture was built to relax the permutation equivariance assumption in GNNs. In the following theorem, we demonstrate that POGNNs relax the permutation equivariance assumption because they are equivariant to the partial order automorphisms, but not necessarily to permutations of all nodes.

Theorem 1. *Let $\mathcal{V}$ be the node set of the input graph, $\mathcal{P}$ a partial order on $\mathcal{V}$, and $\pi \in S_n$ a permutation with matrix representation $\boldsymbol{P}_\pi$. Given a POGNN $f_{\theta,\mathcal{P}}(\boldsymbol{X}, \boldsymbol{A})$, where $\boldsymbol{A}_c$ is the transitive closure of $\mathcal{P}$, held fixed as an operator of the architecture. Then:*

(i) if $\pi \in \text{Aut}(\mathcal{P})$, then for all parameters θ and all inputs $(\boldsymbol{X}, \boldsymbol{A})$,

$$f_{\theta,\mathcal{P}} \left(\boldsymbol{P}_\pi \boldsymbol{X},\, \boldsymbol{P}_\pi \boldsymbol{A} \boldsymbol{P}_\pi^{\top} \right) = \boldsymbol{P}_\pi f_{\theta,\mathcal{P}} \left(\boldsymbol{X}, \boldsymbol{A} \right); \tag{7}$$

(ii) if $\pi \notin \text{Aut}(\mathcal{P})$, there exist parameters θ and inputs $(\boldsymbol{X}, \boldsymbol{A})$ for which Equation (7) fails.

Hence, $\text{Aut}(\mathcal{P})$ is the largest subgroup of S_n under which every POGNN built on $\mathcal{P}$ is equivariant.

A formal proof of Theorem 1 is given in Appendix E. Consequently, POGNNs guarantee permutation equivariance for permutations in $\text{Aut}(\mathcal{P})$ and can learn to become permutation equivariant to permutations in $S_n \setminus \text{Aut}(\mathcal{P})$ depending on the actual data's symmetries. As such, they offer a flexible middle ground between fully permutable sets and strictly ordered sequences. Committing to a partial order relaxes the inductive bias of standard GNNs in a structural way with the size of the symmetry group and hence the strength of relaxation directly determined by the chosen partial order. We believe this tunable permutation equivariance better aligns with complex graph structures,

Table 1: Results for graph-level classification tasks. The partial order yielding the best performance with the POGNN architecture is presented below the results and the **first**, **second**, and **third** best are highlighted. In the first and second rows, results for GINE are displayed if edge features are available; otherwise GIN is used. The mean performance and standard deviation are obtained over 5 random seeds.

Metric	**MUTAG** ACC ↑	**ENZYMES** ACC ↑	**PROTEINS** ACC ↑	**OGBG-MolHIV** ROC-AUC ↑	**Peptides-func** AP ↑	**Peptides-struct** MAE ↓
GIN / GINE	73.7 ± 16.8	41.3 ± 4.2	69.1 ± 5.3	76.8 ± 1.0	0.6612 ± 0.0067	0.2473 ± 0.0017
(+) Virtual Node	**78.9 ± 11.8**	46.2 ± 6.8	71.3 ± 3.5	77.1 ± 1.7	0.6576 ± 0.0094	0.2570 ± 0.0029
GatedGCN	73.2 ± 17.0	44.8 ± 5.8	70.3 ± 3.2	76.2 ± 0.9	0.6765 ± 0.0047	0.2477 ± 0.0009
(+) Virtual Node	76.3 ± 19.1	46.3 ± 4.8	68.8 ± 5.0	**79.1 ± 0.9**	**0.6822 ± 0.0052**	**0.2458 ± 0.0006**
GPS	**83.1 ± 8.4**	**60.0 ± 6.7**	**73.0 ± 2.1**	**78.8 ± 1.0**	0.6534 ± 0.0091	0.2509 ± 0.0014
GRIT	**82.5 ± 5.0**	**57.3 ± 10.3**	**73.2 ± 3.8**	78.4 ± 0.5	**0.6988 ± 0.0082**	**0.2460 ± 0.0012**
POGNN	76.9 ± 11.1 electronegativity	**50.2 ± 9.3** sse type	**74.5 ± 4.9** sse type	**80.2 ± 0.9** electronegativity	**0.6786 ± 0.0071** electronegativity	**0.2436 ± 0.0009** electronegativity

providing a mechanism to control the degree of constrained equivariance through the input's partial order.

Complexity. Let d be the hidden dimension and k the width of the poset, i.e., the size of its largest antichain. For a POGCN layer (Equation (5)), the graph branch costs $\mathcal{O}(nd^2 + |\mathcal{E}|d)$. In the DAG branch, the complexity is given by $\mathcal{O}(nd^2 + |\mathcal{E}_\mathcal{D}|d)$ where $|\mathcal{E}_\mathcal{D}| \propto n^2$. However, approximating $\boldsymbol{A}_c$ by $\boldsymbol{A}_r$ ($|\mathcal{E}_\mathcal{D}| \propto k$) in Equation (5) reduces the complexity to $\mathcal{O}(nd^2 + kd)$. With M heads, the cost of the DAG branch is multiplied by M. Therefore, the overall complexity of a POGNN , when $n >> d$, is $\mathcal{O}(n^2 + |\mathcal{E}|)$ with a GCN as the GNN backbone or $\mathcal{O}(n + |\mathcal{E}|)$ for the approximation replacing $\boldsymbol{A}_c$ by $\boldsymbol{A}_r$. The full derivation is given in Appendix F.

6 Experimental Results

We validate our theoretical insights against GNNs and Graph Transformers on biochemical graph-level tasks from TUDataset (Morris et al., 2020), OGB (Hu et al., 2020a) and LRGB (Dwivedi et al., 2022). Details on datasets, baselines and setup are given in Appendix G and code is provided in the Supplementary Material.

6.1 Partial Order Construction

The partial order $\mathcal{P}$ inducing the DAG is a central design choice. We define chemically motivated orders for three types of graphs.

Atom-level molecular graphs. (1) Atomic number, a proxy for chemical role separating heavier from lighter atoms; (2) electronegativity, obtained by binning Pauling values, which govern bond polarity and partial charges, hence where a molecule reacts; (3) a canonical order given by Morgan invariants (Morgan, 1965), which iteratively refine atom invariants by aggregating those of their neighbours, extracted with RDKit and binned into B levels.

SSE-level protein graphs. (4) SSE type: turn $\prec$ helix and turn $\prec$ sheet, leaving helices and sheets incomparable, as turns are short irregular linkers subordinate to the hydrogen-bonded elements forming the fold; (5) polarity, binned into three levels following Borgwardt et al. (2005), separating apolar core elements from polar and charged surface elements.

Atom-level peptide graphs. Besides orders (1)-(3), which are relevant since electronic interactions shape protein structure and function, we use (6) a peptide-chain order following the N-to-C direction: each atom takes the binned index of its residue, so atoms in the same or consecutive residues are incomparable while distant residues are ordered.

6.2 Graph-Level Tasks

As shown in Table 1, POGNN outperforms GIN/GINE and GatedGCN on all six datasets and is best overall on PROTEINS, OGBG-MolHIV and Peptides-struct. It ranks third on ENZYMES,

Table 2: Comparison of POGNN with other approaches to relaxing permutation equivariance (Lawrence et al., 2025; Huang et al., 2023). The mean and standard deviation are obtained over 5 random seeds. The **first** and <u>second</u> results are highlighted.

Metric	**ENZYMES** ACC ↑	**OGBG-MolHIV** ROC-AUC ↑
GIN / GINE + SymPE	30.0 ± 7.2	64.2 ± 5.0
GatedGCN + SymPE	34.7 ± 7.1	<u>74.5 ± 3.7</u>
GPS + SymPE	33.6 ± 4.0	63.8 ± 2.8
AEGN	**50.2 ± 7.0**	69.0 ± 5.4
POGNN	**50.2 ± 9.3**	**80.2 ± 0.9**

behind both Graph Transformers, while improving over the strongest message-passing model and its virtual-node variants, and third on Peptides-func, behind GRIT and GatedGCN + VN.

Beyond long-range shortcuts. The DAG branch could merely add shortcuts between distant nodes. Virtual nodes provide exactly such global interactions (Southern et al., 2025), yet POGNN outperforms them on four of six datasets, consistent with gains stemming from the order-induced inductive bias rather than from connectivity alone.

Choice of partial order. For each dataset, we train POGNN with every candidate order over five seeds and report the one with the best mean test performance. Electronegativity is best on all atom-level datasets and SSE type on both protein datasets. On MUTAG, even the best order leaves POGNN behind GIN + VN and both Graph Transformers, suggesting that none of our orders suits mutagenicity prediction: the benefit of POGNN depends on an order aligned with the task.

Alternative symmetry relaxations. Table 2 compares POGNN with SymPE (Lawrence et al., 2025) which breaks symmetry at the input through a sampled positional encoding, and AEGN (Huang et al., 2023), which, like POGNN, restricts the symmetry group in the architecture. The architectural approach appears more effective: AEGN outperforms SymPE on ENZYMES and two of its three variants on OGBG-MolHIV, while POGNN is best on OGBG-MolHIV, ahead of GatedGCN + SymPE and AEGN, and ties with AEGN on ENZYMES. SymPE even degrades its backbones, plausibly because its symmetry-breaking input is resampled at every forward pass, whereas POGNN commits to a single group $\mathrm{Aut}(\mathcal{P})$. This supports breaking symmetry structurally, via a DAG-based partial order, rather than through input features.

Alternative DAG layers. Figure 7 in Appendix G.4 replaces our DAG convolution with DAGNN (Thost & Chen, 2021) or DAGformer (Luo et al., 2023), keeping the rest fixed. Our layer is best in three of six configurations, including those of Table 1 (SSE type on ENZYMES, electronegativity on OGBG-MolHIV). Our layer is thus the most reliable across orders, matching or outperforming DAGformer without global attention over reachable pairs.

7 Conclusion

In Section 4, we introduced Partially Ordered Graphs, which augment a graph with a node partial order encoded as a DAG. We also proposed POGNNs (Section 5), which combine graph and DAG convolutions to learn functions on POGs. Also, in the same section, we proved that POGNNs are equivariant to the poset automorphisms, so the choice of partial order tunes the equivariance constraint. Finally, we showed in Section 6 that POGNNs improve empirically over GNNs on most benchmarks considered and over GTs on half of them. It also outperforms other approaches to relaxing permutation equivariance, motivating the structural symmetry breaking via a DAG learning approach.

Future Work. The performance gains from POGNN depend on choosing an order suited to the task. Our partial orders are currently defined heuristically from domain knowledge. Learning them end-to-end is a promising direction that would remove this dependence.

REFERENCES

S Kamal Abdali and B David Saunders. Transitive Closure and Related Semiring Properties via Eliminants. *Theoretical Computer Science*, 1985.

Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alexander J. Smola, and Hans-Peter Kriegel. Protein Function Prediction via Graph Kernels. *Proceedings of the 13th International Conference on Intelligent Systems for Molecular Biology*, 2005.

Fedor Borisyuk, Shihai He, Yunbo Ouyang, Morteza Ramezani, Peng Du, Xiaochen Hou, Chengming Jiang, Nitin Pasumarthy, Priya Bannur, Birjodh Tiwana, et al. LIGNN: Graph Neural Networks at LinkedIn. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024.

Xavier Bresson and Thomas Laurent. Residual Gated Graph Convnets. *arXiv:1711.07553*, 2017.

Pim de Haan, Taco S Cohen, and Max Welling. Natural Graph Networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Marco De Nadai, Francesco Fabbri, Paul Gigioli, Alice Wang, Ang Li, Fabrizio Silvestri, Laura Kim, Shawn Lin, Vladan Radosavljevic, Sandeep Ghael, et al. Personalized Audiobook Recommendations at Spotify through Graph Neural Networks. In *Proceedings of the ACM Web Conference*, 2024.

Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. ETA Prediction with Graph Neural Networks in Google Maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long Range Graph Benchmark. *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Nadav Dym, Hannah Lawrence, and Jonathan W. Siegel. Equivariant Frames and the Impossibility of Continuous Canonicalization. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A Fair Comparison of Graph Neural Networks for Graph Classification. *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2020a.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for Pre-Training Graph Neural Networks. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020b.

Ningyuan Huang, Ron Levie, and Soledad Villar. Approximately Equivariant Graph Networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Sékou-Oumar Kaba and Siamak Ravanbakhsh. Symmetry Breaking and Equivariant Neural Networks. *arXiv:2312.09016*, 2023.

Arthur B Kahn. Topological Sorting of Large Networks. *Communications of the ACM*, 1962.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Hannah Lawrence, Vasco Portilheiro, Yan Zhang, and Sékou-Oumar Kaba. Improving Equivariant Networks with Probabilistic Symmetry Breaking. *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 2002.

Yuankai Luo, Veronika Thost, and Lei Shi. Transformers over Directed Acyclic Graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yuankai Luo, Lei Shi, and Xiao-Ming Wu. Can Classic Gnns Be Strong Baselines for Graph-level Tasks? Simple Architectures Meet Excellence. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, Puneet K Dokania, Mark Coates, Philip Torr, and Ser-Nam Lim. Graph Inductive Biases in Transformers Without Message Passing. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

Harry L Morgan. The Generation of a Unique Machine Description for Chemical Structures- A Technique Developed at Chemical Abstracts Service. *Journal of chemical documentation*, 1965.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Samuel Rey, Hamed Ajorlou, and Gonzalo Mateos. Directed Acyclic Graph Convolutional Networks. *IEEE Transactions on Signal Processing*, 2026.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Bastian Seifert, Chris Wendler, and Markus Püschel. Causal Fourier Analysis on Directed Acyclic Graphs and Posets. *IEEE Transactions on Signal Processing*, 2023.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 2008.

Joshua Southern, Francesco Di Giovanni, Michael Bronstein, and Johannes F Lutzeyer. Understanding Virtual Nodes: Oversquashing and Node Heterogeneity. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Veronika Thost and Jie Chen. Directed Acyclic Graph Neural Networks. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.

Petar Veličković. Everything is Connected: Graph Neural Networks. *Current Opinion in Structural Biology*, 2023.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph Attention Networks. In *Proceedings of 6th International Conference on Learning Representations (ICLR)*, 2018.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

Appendix

Table of Contents

A Relaxing Equivariance in GNNs **13**

B Transitive Closure in Directed Acyclic Graphs **13**

C An Example of a Partially Ordered Graph **15**

D Convolution by a Bank of Learnable Filters on DAGs **15**

E Proof of Theorem 1 **16**

F Complexity Analysis of POGNNs **17**

G Experiments Details **18**

G.1 Datasets 18

G.2 Baselines 18

G.3 Experimental Setup 19

G.4 Ablation Study 19

A Relaxing Equivariance in GNNs

This section provides motivation for our approach. Firstly, strictly permutation equivariant models like GNNs need to permute with all possible permutations of the input graph. However, as noted in a recent survey (Veličković, 2023), the permutation equivariance assumption is often not the right inductive bias for every graph and nodes labels might not be all interchangeable. Therefore, the permutation equivariance assumption does not always match the actual data's symmetries (see Figure 1).

Therefore, the goal of the proposed architecture is to target the intermediate regime of node posets as permutation equivariance in GNNs stems from the fact that graphs are nodes sets. Sequence models working with node sequences do not account for any symmetry in the input graph. Moreover, designing a model that accounts for fewer symmetries reduces the number of constraints and should increase the size of the hypothesis function class (see Figure 3). These arguments motivate our choice to enhance graphs with node posets in order to relax the permutation equivariance found in GNNs to more accurately match the input data's symmetries.

B Transitive Closure in Directed Acyclic Graphs

Figure 4 illustrates the structural differences between the transitively reduced and closed versions of a DAG. This influence is captured by the weighted transitive closure, which formalises how signals propagate through the DAG hierarchy (from left to right). As shown in Figure 5, this process relies on two fundamental algebraic operations: a product operation $\odot$ to combine weights along a single path, and a summation operation $\oplus$ to aggregate the contributions from multiple parallel paths.

While the weight transitive closure defined in the main text of our manuscript is tailored to our construction, alternative definitions exist depending on the underlying semiring structure chosen for the path aggregation. Table 3 provides several examples of such closures, adapted from Abdali & Saunders (1985); Seifert et al. (2023), illustrating how different choices of $\oplus$ and $\odot$ operations lead to distinct interpretations.

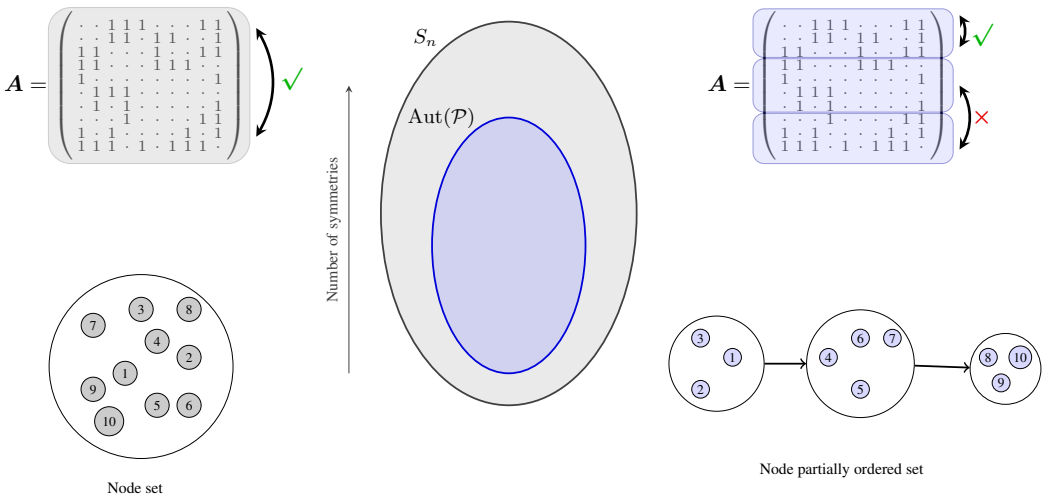

Figure 3: GNNs work with the node set as input and hence need to permute with all the possible permutations in the set, inducing a large number of symmetries ($n!$ for a graph with n nodes). However, when working with a node poset the model needs only to permute with the permutations in the automorphism of the poset, reducing the number of symmetries it must account for. What is more, the strength of relaxation or number of symmetries depends on the input partial order allowing a tunable relaxation of permutation equivariance.

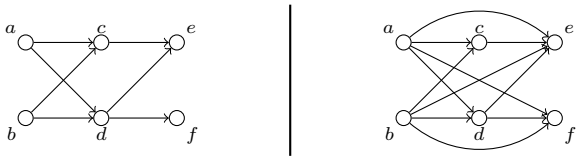

Figure 4: Illustration of the difference between a reduced version of a DAG on the **(Left)** and its closed version on the **(Right)**. Both DAGs represent the same underlying poset (adapted from Seifert et al. (2023)).

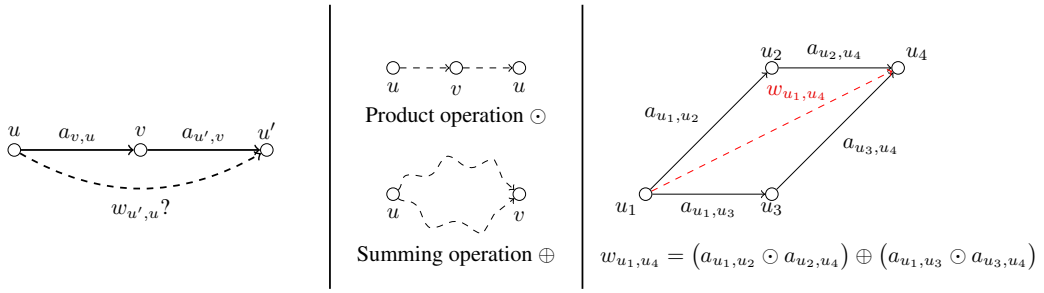

Figure 5: Illustration of the weighted transitive closure on DAGs (adapted from Seifert et al. (2023)). **(Left)** Weighted transitive closure problem for a simple DAG with three nodes and two edges. **(Middle)** The two operations defining the closure: the product $\odot$ combining consecutive weights along a path and the summation $\oplus$ combining weights over alternative paths. **(Right)** An example with 4 nodes u_1, u_2, u_3, u_4 where the transitive weight w_{u_1,u_4} is computed as the sum of products along two distinct paths.

Table 3: Examples of choices for weight transitive closure. Here $\mathcal{S}$ denotes a closed set which in our case represents the edge weights, $0_{\mathcal{S}} \in \mathcal{S}$ is the identity element of $\oplus$, and $1_{\mathcal{S}} \in \mathcal{S}$ is the identity element of $\odot$.

$\mathcal{S}$	$u \oplus v$	$u \odot v$	$0_{\mathcal{S}}$	$1_{\mathcal{S}}$	Description
$\{0, 1\}$	u or v	u and v	0	1	Reachability in graphs (Boolean values)
$[0, 1]$	$u + v$	uv	0	1	Fraction of pollution from v reaching u
$[0, 1]$	$\max(u, v)$	uv	0	1	Most reliable path from v to u
$\mathbb{R}^+ \cup \{\infty\}$	$\min(u, v)$	$u + v$	∞	0	Shortest path length from v to u
$\mathbb{R}^+ \cup \{\infty\}$	$\max(u, v)$	$\min(u, v)$	0	∞	Largest capacity from v to u

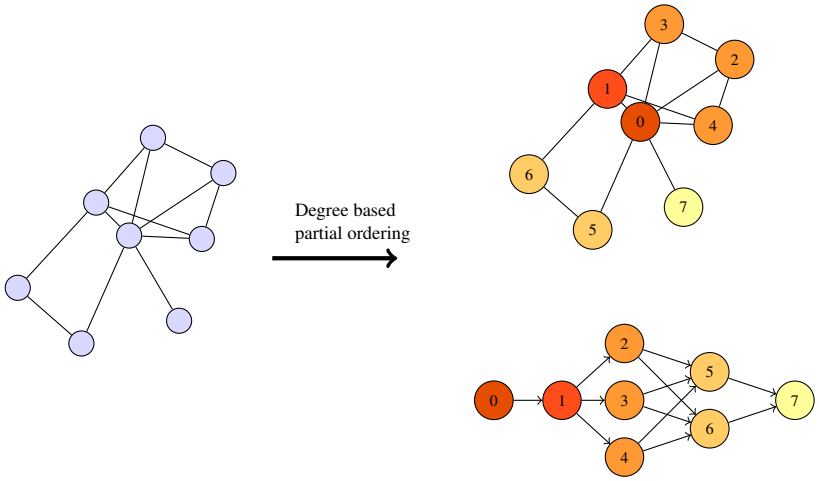

Figure 6: POG Construction: **Left**: Input graph $\mathcal{G}$. **Upper Right**: The input graph relabeled according to a linear extension of the node poset. **Lower Right**: The resulting DAG $\mathcal{D}$ induced by the partial order, where edges are oriented to strictly follow the relative node comparisons.

C An Example of a Partially Ordered Graph

Figure 6 illustrates the construction of a POG from an input graph $\mathcal{G}$ and a degree-based node partial order. We first materialise the partial order as a DAG $\mathcal{D} = (\mathcal{V}, \mathcal{E}_{\mathcal{D}})$ whose directed edges encode all comparisons $u \prec v$, then we use the DAG to infer node indices that preserve the degree-based partial order using the Kahn algorithm, and finally we relabel all the nodes in the input graph according to these new node indices.

D Convolution by a Bank of Learnable Filters on DAGs

The causal graph shift operators on DAGs, central to defining convolutions on DAGs, are introduced in Seifert et al. (2023). They define one shift operator $\boldsymbol{S}_u$ per node. This operator $\boldsymbol{S}_u$ is idempotent, and since shifts generate the algebra (ring and vector space) of filters, the convolution of a signal $\boldsymbol{X}$ by a filter $\boldsymbol{\Theta} = (\theta_u)_{u \in \mathcal{V}} \in \mathbb{R}^n$

$$\boldsymbol{X}_{\text{conv}} = \sum_{u \in \mathcal{V}} \theta_u \boldsymbol{S}_u \boldsymbol{X},$$

also from Seifert et al. (2023), we know

$$\boldsymbol{S}_u \boldsymbol{X} = \boldsymbol{A}_c \boldsymbol{\Gamma}_u c,$$

where c is the cause of the signal x given by $c = \boldsymbol{A}_c^{-1}x$ and $\boldsymbol{\Gamma}_u$ is the diagonal matrix such that $(\boldsymbol{\Gamma}_u)_{i,i} = \mathbb{1}_{i \preceq u}$. Hence, we can deduce

$$\begin{aligned}\boldsymbol{X}_{\text{conv}} &= \sum_{u \in \mathcal{V}} \theta_u \boldsymbol{A}_c \boldsymbol{\Gamma}_u \boldsymbol{A}_c^{-1} \boldsymbol{X} \\ &= \boldsymbol{A}_c \Big(\sum_{u \in \mathcal{V}} \theta_u \boldsymbol{\Gamma}_u \Big) \boldsymbol{A}_c^{-1} \boldsymbol{X} \\ &= \boldsymbol{A}_c \boldsymbol{D} \boldsymbol{A}_c^{-1} \boldsymbol{X},\end{aligned}$$

with $\boldsymbol{D}$ the diagonal matrix corresponding to the filter coefficients such that $\boldsymbol{D}_{i,i} = \sum_{k=1}^{n} \delta_{i \preceq k} \theta_k$ with $\preceq$ induced by the partial order $\mathcal{P}$. Then, by changing variables such that $\theta'_n = \theta_n + \theta_{n-1}$ and $\theta'_i = \theta'_{i-1} - \theta_i$, we have $\mathbf{D}_{i,i} = \theta'_i$. Finally, the choice of transitive closure in this paper $\boldsymbol{A}_c = (\boldsymbol{I} - \boldsymbol{A}_r)^{-1}$ implies

$$\boldsymbol{X}_{\text{conv}} = \boldsymbol{A}_c \boldsymbol{D} (\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{X}.$$

E Proof of Theorem 1

Lemma 1. *For $\pi \in S_n$, the following are equivalent:*

(a) $\pi \in \mathrm{Aut}(\mathcal{P})$, *i.e.,* $u \prec v \iff \pi(u) \prec \pi(v)$,

(b) $\boldsymbol{P}_\pi \boldsymbol{A}_r \boldsymbol{P}_\pi^\top = \boldsymbol{A}_r$,

(c) $\boldsymbol{P}_\pi \boldsymbol{A}_c \boldsymbol{P}_\pi^\top = \boldsymbol{A}_c$.

Proof of Lemma 1. (a) $\Rightarrow$ (b). First, by definition $(\boldsymbol{P}_\pi \boldsymbol{A}_r \boldsymbol{P}_\pi^\top)_{\pi(u)\pi(v)} = (\boldsymbol{A}_r)_{uv}$. Besides, the cover relation is defined from $\prec$ alone, so an order automorphism preserves it: $u \prec\cdot\, v \iff \pi(u) \prec\cdot\, \pi(v)$. Hence $(\boldsymbol{A}_r)_{\pi(u)\pi(v)} = (\boldsymbol{A}_r)_{uv} = (\boldsymbol{P}_\pi \boldsymbol{A}_r \boldsymbol{P}_\pi^\top)_{\pi(u)\pi(v)}$ for all u, v.

(b) $\Rightarrow$ (c). Since $\boldsymbol{P}_\pi^\top = \boldsymbol{P}_\pi^{-1}$, we have $(\boldsymbol{I} - \boldsymbol{P}_\pi \boldsymbol{A}_r \boldsymbol{P}_\pi^\top)^{-1} = \boldsymbol{P}_\pi (\boldsymbol{I} - \boldsymbol{A}_r)^{-1} \boldsymbol{P}_\pi^\top$. Therefore, with our choice of transitive closure $\boldsymbol{A}_c = (\boldsymbol{I} - \boldsymbol{A}_r)^{-1}$, we have $\boldsymbol{A}_c = \boldsymbol{P}_\pi \boldsymbol{A}_c \boldsymbol{P}_\pi^\top$.

(c) $\Rightarrow$ (a). Assuming (c) gives $(\boldsymbol{A}_c)_{\pi(u)\pi(v)} = (\boldsymbol{A}_c)_{uv}$. Furthermore, the entry $(\boldsymbol{A}_r^k)_{uv}$ counts the directed paths of length k from u to v in the cover DAG. Hence $(\boldsymbol{A}_c)_{uv}$ is the total number of such paths, which is positive if and only if $u \preceq v$. What follows is $(\boldsymbol{A}_c)_{\pi(u)\pi(v)} = 1 \iff (\boldsymbol{A}_c)_{uv} = 1$ and $u \preceq v \iff \pi(u) \preceq \pi(v)$. □

Proof of Theorem 1. **(i)** Let $\pi \in \mathrm{Aut}(\mathcal{P})$.

Graph branch: Standard GNN layers are equivariant to all of S_n, so

$$\mathrm{GNN}^{(\ell)}(\boldsymbol{P}_\pi \boldsymbol{H}^{(\ell)}, \boldsymbol{P}_\pi \boldsymbol{A} \boldsymbol{P}_\pi^\top) = \boldsymbol{P}_\pi \,\mathrm{GNN}^{(\ell)}(\boldsymbol{H}^{(\ell)}, \boldsymbol{A}).$$

DAG branch: By Lemma 1, $\boldsymbol{A}_c = \boldsymbol{P}_\pi \boldsymbol{A}_c \boldsymbol{P}_\pi^\top$ and $\boldsymbol{A}_r = \boldsymbol{P}_\pi \boldsymbol{A}_r \boldsymbol{P}_\pi^\top$. Using $(\boldsymbol{P}_\pi \mathbf{U}) \odot (\boldsymbol{P}_\pi \mathbf{V}) = \boldsymbol{P}_\pi (\mathbf{U} \odot \mathbf{V})$, we obtain

$$\begin{aligned} f_{\theta,\mathcal{P}}\left(\boldsymbol{P}_\pi \boldsymbol{X},\, \boldsymbol{P}_\pi \boldsymbol{A} \boldsymbol{P}_\pi^\top\right) &= \boldsymbol{A}_c \left[(\boldsymbol{P}_\pi \boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)}) \odot \left((\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{P}_\pi \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)} \right) \right] \\ &= \boldsymbol{P}_\pi \boldsymbol{A}_c \boldsymbol{P}_\pi^\top \left[(\boldsymbol{P}_\pi \boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)}) \odot \left(\boldsymbol{P}_\pi (\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)} \right) \right] \\ &= \boldsymbol{P}_\pi \boldsymbol{A}_c \boldsymbol{P}_\pi^\top \left[\boldsymbol{P}_\pi (\boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)}) \odot \left((\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)} \right) \right] \\ &= \boldsymbol{P}_\pi \, \boldsymbol{A}_c \left[(\boldsymbol{H}^{(\ell)} \boldsymbol{\Theta}_{\mathcal{D}}^{(\ell)}) \odot \left((\boldsymbol{I} - \boldsymbol{A}_r) \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell)} \right) \right]. \end{aligned}$$

The aggregation $\oplus$ (sum, mean or concatenation) and the activation σ act row-wise, so both commute with $\boldsymbol{P}_\pi$. Therefore,

$$f_{\theta,\mathcal{P}}\left(\boldsymbol{P}_\pi \boldsymbol{X},\, \boldsymbol{P}_\pi \boldsymbol{A} \boldsymbol{P}_\pi^\top\right) = \boldsymbol{P}_\pi f_{\theta,\mathcal{P}}\left(\boldsymbol{X}, \boldsymbol{A}\right).$$

(ii) Let $\pi \notin \mathrm{Aut}(\mathcal{P})$. We construct a one-layer POGNN that is not equivariant to π. Set $d_{\text{in}} = 2$, the graph branch to zero ($\boldsymbol{W}_{\mathcal{G}} = \mathbf{0}$), take one output channel, and choose $\boldsymbol{\Theta}_{\mathcal{D}} = \mathbf{e}_1 = [1, 0]^\top \in \mathbb{R}^{d_{\text{in}} \times 1}$ and $\boldsymbol{W}_{\mathcal{D}} = \mathbf{e}_2 = [0, 1]^\top \in \mathbb{R}^{d_{\text{in}} \times 1}$. Take $\boldsymbol{X} = [\mathbf{a}, \mathbf{b}] \in \mathbb{R}^{n \times 2}$, we have $\boldsymbol{X}\boldsymbol{\Theta}_{\mathcal{D}} = \mathbf{a}$ and $\boldsymbol{X}\boldsymbol{W}_{\mathcal{D}} = \mathbf{b}$. Then, Equation (4) reduces to

$$f(\mathbf{a}, \mathbf{b}) = \sigma\big(\mathbf{Z}(\mathbf{a}, \mathbf{b})\big), \qquad \mathbf{Z}(\mathbf{a}, \mathbf{b}) = \boldsymbol{A}_c \,\mathrm{diag}(\mathbf{a}) \boldsymbol{A}_c^{-1} \mathbf{b} \in \mathbb{R}^n,$$

where we used $\mathbf{a} \odot (\boldsymbol{A}_c^{-1}\mathbf{b}) = \mathrm{diag}(\mathbf{a})\boldsymbol{A}_c^{-1}\mathbf{b}$ and σ acts entry-wise.

Suppose, for contradiction, that this POGNN is π-equivariant for every input, i.e., $f(\boldsymbol{P}_\pi \mathbf{a}, \boldsymbol{P}_\pi \mathbf{b}) = \boldsymbol{P}_\pi f(\mathbf{a}, \mathbf{b})$ for all $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$.

Since σ acts entry-wise,

$$\mathbf{Z}(\boldsymbol{P}_\pi \mathbf{a}, \boldsymbol{P}_\pi \mathbf{b}) = \boldsymbol{P}_\pi \, \mathbf{Z}(\mathbf{a}, \mathbf{b}) \qquad \text{for all } \mathbf{a}, \mathbf{b} \in \mathbb{R}^n. \tag{8}$$

By definition, $(\boldsymbol{P}_\pi)_{\pi(u),u} = 1$ so $\mathrm{diag}(\boldsymbol{P}_\pi \mathbf{a}) = \boldsymbol{P}_\pi \,\mathrm{diag}(\mathbf{a}) \boldsymbol{P}_\pi^\top$. Expanding both sides of Equation 8,

$$\boldsymbol{A}_c \boldsymbol{P}_\pi \,\mathrm{diag}(\mathbf{a}) \boldsymbol{P}_\pi^\top \boldsymbol{A}_c^{-1} \boldsymbol{P}_\pi \,\mathbf{b} = \boldsymbol{P}_\pi \boldsymbol{A}_c \,\mathrm{diag}(\mathbf{a}) \boldsymbol{A}_c^{-1} \mathbf{b}.$$

Left-multiplying by $\boldsymbol{P}_\pi^\top = \boldsymbol{P}_\pi^{-1}$ and using $\boldsymbol{P}_\pi \boldsymbol{P}_\pi^\top = \boldsymbol{I}$ yields

$$\big(\boldsymbol{P}_\pi^\top \boldsymbol{A}_c \boldsymbol{P}_\pi\big) \,\mathrm{diag}(\mathbf{a}) \big(\boldsymbol{P}_\pi^\top \boldsymbol{A}_c^{-1} \boldsymbol{P}_\pi\big) \mathbf{b} = \boldsymbol{A}_c \,\mathrm{diag}(\mathbf{a}) \boldsymbol{A}_c^{-1} \mathbf{b}.$$

Set $\boldsymbol{A}_c' = \boldsymbol{P}_\pi^\top \boldsymbol{A}_c \boldsymbol{P}_\pi$. The identity above therefore reads

$$\boldsymbol{A}_c' \,\mathrm{diag}(\mathbf{a}) \boldsymbol{A}_c'^{-1} \mathbf{b} = \boldsymbol{A}_c \,\mathrm{diag}(\mathbf{a}) \boldsymbol{A}_c^{-1} \mathbf{b} \qquad \text{for all } \mathbf{a}, \mathbf{b} \in \mathbb{R}^n. \tag{9}$$

Equation 9 is an equality of vectors. Fixing $\mathbf{a}$ and letting $\mathbf{b}$ range over the canonical basis $\mathbf{e}_1, \ldots, \mathbf{e}_n$ shows that the two matrices agree column by column, hence

$$\boldsymbol{A}_c' \,\mathrm{diag}(\mathbf{a}) \boldsymbol{A}_c'^{-1} = \boldsymbol{A}_c \,\mathrm{diag}(\mathbf{a}) \boldsymbol{A}_c^{-1} \qquad \text{for all } \mathbf{a} \in \mathbb{R}^n. \tag{10}$$

Set $\mathbf{N} = \boldsymbol{A}_c^{-1} \boldsymbol{A}_c'$. Multiplying Equation 10 on the left by $\boldsymbol{A}_c^{-1}$ and on the right by $\boldsymbol{A}_c'$ yields

$$\mathbf{N} \,\mathrm{diag}(\mathbf{a}) = \mathrm{diag}(\mathbf{a}) \mathbf{N} \qquad \text{for all } \mathbf{a} \in \mathbb{R}^n.$$

Fix any $\mathbf{a}$ with pairwise distinct entries, e.g. $\mathbf{a} = (1, 2, \ldots, n)^\top$. Reading off entry (i, j) of both sides,

$$\big(\mathbf{N} \,\mathrm{diag}(\mathbf{a})\big)_{ij} = \mathbf{N}_{ij} \mathbf{a}_j, \qquad \big(\,\mathrm{diag}(\mathbf{a}) \mathbf{N}\big)_{ij} = \mathbf{a}_i \mathbf{N}_{ij},$$

so $(\mathbf{a}_j - \mathbf{a}_i)\mathbf{N}_{ij} = 0$ for all i, j. For $i \neq j$ we have $\mathbf{a}_i \neq \mathbf{a}_j$, hence $\mathbf{N}_{ij} = 0$. Thus $\mathbf{N}$ is diagonal.

Every diagonal entry of $\boldsymbol{A}_c$ equals 1 by convention, and hence the same holds for $\boldsymbol{A}_c'$. From $\boldsymbol{A}_c' = \boldsymbol{A}_c \mathbf{N}$ with $\mathbf{N}$ diagonal, $(\boldsymbol{A}_c')_{uu} = (\boldsymbol{A}_c)_{uu} (\mathbf{N})_{uu}$, so $\mathbf{N} = \boldsymbol{I}$. Thus $\boldsymbol{P}_\pi^\top \boldsymbol{A}_c \boldsymbol{P}_\pi = \boldsymbol{A}_c$, equivalently $\boldsymbol{P}_\pi \boldsymbol{A}_c \boldsymbol{P}_\pi^\top = \boldsymbol{A}_c$. By Lemma 1, $\pi \in \mathrm{Aut}(\mathcal{P})$, contradicting the assumption. Hence no such π can leave every POGNN equivariant, which proves (ii). □

Finally, $\mathrm{Aut}(\mathcal{P})$ is a group. By (i), every POGNN is equivariant to it, and by (ii), no permutation outside it is guaranteed to be a symmetry of every POGNN built on $\mathcal{P}$. It is therefore the largest such subgroup.

F Complexity Analysis of POGNNs

For the complexity analysis of a POGNN layer, we assume for all ℓ, $d_{\text{in}}^{(\ell)} = d_{\text{out}}^{(\ell)} = d$. As evident from Equation (4), a POGNN involves a graph layer and a DAG convolution.

For the special case of POGCN in Equation (5), the complexity of the graph convolution is given by the complexity of matrix multiplication $\mathbf{Z}^{(\ell)} = \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{G}}^{(\ell+1)}$ which is $\mathcal{O}(nd^2)$ and by the complexity of matrix multiplication with $\widetilde{\boldsymbol{D}}^{-1/2} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-1/2} \mathbf{Z}^{(\ell)}$ (neighborhood aggregation). Since $\widetilde{\boldsymbol{D}}^{-1/2} \widetilde{\boldsymbol{A}} \widetilde{\boldsymbol{D}}^{-1/2}$ is a sparse matrix, the complexity of the overall graph convolution is $\mathcal{O}(nd^2 + |\mathcal{E}|d)$.

For the DAG convolution, the complexity analysis can be broken down into several parts. First, computing $\mathbf{Z}^{(\ell)} = \boldsymbol{H}^{(\ell)} \boldsymbol{W}_{\mathcal{D}}^{(\ell+1)}$ is a dense matrix multiplication hence its complexity is $\mathcal{O}(nd^2)$.

Then, computing $\hat{\mathbf{Z}}^{(\ell)} = (\boldsymbol{I} - \boldsymbol{A}_r)\mathbf{Z}^{(\ell)}$ is a sparse matrix multiplication and its complexity is given by $\mathcal{O}(|\mathcal{E}_{\mathcal{D}}|d)$. Next, computing $\hat{\mathbf{Z}}^{(\ell)}_{\text{conv}} = \boldsymbol{\Theta}^{(\ell+1)}_{\mathcal{D}} \odot \hat{\mathbf{Z}}^{(\ell)}$ is an element-wise operation hence its complexity is $\mathcal{O}(nd)$. Finally, computing $\mathbf{Z}^{(\ell)}_{\text{conv}} = \boldsymbol{A}_c\hat{\mathbf{Z}}^{(\ell)}_{\text{conv}}$ is a multiplication between a triangular matrix and a dense matrix that represents a neighborhood aggregation operation, and its complexity is $\mathcal{O}(|\mathcal{E}_{\mathcal{D}}|d)$. In transitively closed DAGs, we often have $|\mathcal{E}_{\mathcal{D}}| \propto n^2$. Overall, the complexity of the DAG convolution layer is $\mathcal{O}(nd^2 + |\mathcal{E}_{\mathcal{D}}|d + nd + |\mathcal{E}_{\mathcal{D}}|d) = \mathcal{O}(nd^2 + |\mathcal{E}_{\mathcal{D}}|d) = \mathcal{O}(nd^2 + n^2d)$.

A more efficient, DAG convolution layer can be built by approximating the transitively closed adjacency matrix $\boldsymbol{A}_c$ with the transitively reduced adjacency matrix $\boldsymbol{A}_r$. Then, the complexity of the corresponding operation of the DAG convolution layer, computing $\mathbf{Z}^{(\ell)}_{\text{conv}} = \boldsymbol{A}_r\hat{\mathbf{Z}}^{(\ell)}_{\text{conv}}$, is $\mathcal{O}\left(|\mathcal{E}_{\mathcal{D}}|d\right)$ but in a transitively reduced DAG $|\mathcal{E}_{\mathcal{D}}| \propto k$, with k being the width of the DAG,i.e., the longest antichain or maximal number of mutually non-comparable elements of the DAG, rather than $|\mathcal{E}_{\mathcal{D}}| \propto n^2$ for transitively closed DAGs.

The complexity of the POGCN architecture is, hence, given by $\mathcal{O}(nd^2 + |\mathcal{E}_{\mathcal{D}}|d + nd^2 + |\mathcal{E}|d)$. What is more, in practice $n \gg d$, k because having $n \approx k$ would mean that almost every node in the DAG is not comparable and that the partial order does not bring much information on the underlying structure of the graph, making the POGCN architecture irrelevant compared to a traditional GNN. Under these assumptions, the complexity of the POGCN reduces to $\mathcal{O}(n + |\mathcal{E}_{\mathcal{D}}| + |\mathcal{E}|)$. When using the transitively closed version of the DAG, we have $|\mathcal{E}_{\mathcal{D}}| \propto n^2$ and the complexity of the POGCN is $\mathcal{O}(n^2 + |\mathcal{E}|)$, and when using the transitively reduced version of the DAG, we have $|\mathcal{E}_{\mathcal{D}}| \propto k$ and the complexity of the POGCN is $\mathcal{O}(n + |\mathcal{E}|)$ which is similar to the complexity of a traditional GNN.

G Experiments Details

This section gathers the implementation details of the experiments discussed in Section 6, including dataset statistics and splits, baseline configurations, hardware, and the hyperparameter tuning strategy.

G.1 Datasets

To evaluate our approach on graph-level tasks, we use a collection of widely used biochemical benchmarks. We conduct experiments on the datasets MUTAG, PROTEINS and ENZYMES from the TU Dataset (Morris et al., 2020), OGBG-MolHIV from the Open Graph Benchmark (Hu et al., 2020a), and Peptides-func and Peptides-struct from the Long Range Graph Benchmark (Dwivedi et al., 2022).

Table 4: Summary statistics of graph-level datasets used in the experiments.

Dataset	Number of Graphs	Avg Number of Nodes	Avg Number of Edges	Features	Task
MUTAG	188	17.9	19.7	7	Binary classification
ENZYMES	600	32.6	62.1	18	Multi-class classification
PROTEINS	1113	39.1	72.8	29	Binary classification
OGBG-MolHIV	41,127	25.5	27.5	9	Binary classification
Peptides-func	10,873	151.5	308.5	9	Multi-label classification
Peptides-struct	10,873	151.5	308.5	9	Graph regression

For dataset split strategies, we use the provided train/validation/testing splits provided for all the datasets except the ones from the TU datasets. For MUTAG, PROTEINS, and ENZYMES, we use the strategy described in (Errica et al., 2020).

G.2 Baselines

We evaluate our approach against various baseline approaches, which can be split into two categories: Graph Neural Networks (GatedGCN Bresson & Laurent (2017), and GIN Xu et al. (2019); Hu et al. (2020b)) and GraphTransformers (GraphGPS Rampášek et al. (2022), GRIT Ma et al. (2023)). To complete our analysis, for each dataset, we add a virtual node each GNN variant, as described in Southern et al. (2025), to show that the improved performance of our approach does not simply benefit from long-range interactions captured in the DAG but rather from a change in inductive bias.

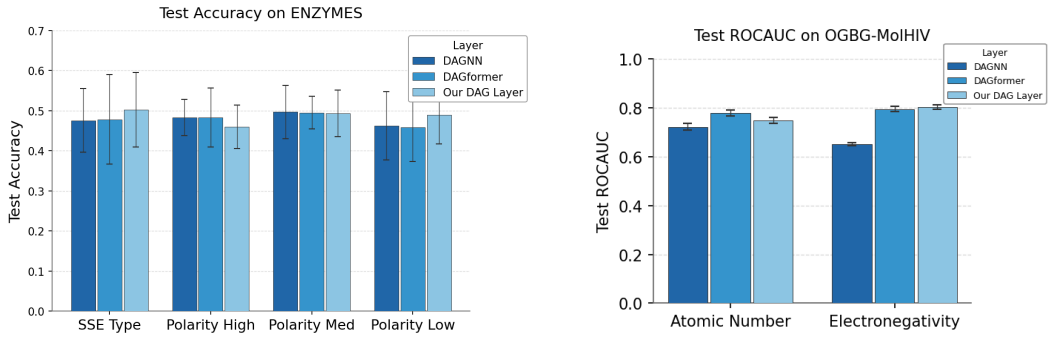

Figure 7: Comparison of DAG layers within POGNN. The DAG convolution of POGNN is replaced by DAGNN (Thost & Chen, 2021) or DAGformer (Luo et al., 2023), keeping the rest of the architecture fixed. (Left) Test accuracy on ENZYMES with the SSE-type order and three variants of the polarity order. (Right) Test ROC-AUC on OGBG-MolHIV with the atomic number and electronegativity orders. Error bars denote the standard deviation over 5 folds on ENZYMES and over 5 random seeds on OGBG-MolHIV.

G.3 Experimental Setup

Configurations. The experiments were performed using either a single NVIDIA RTX Pro 6000 Blackwell GPU or a single NVIDIA H100 GPU on a high-performance computing cluster. For all the datasets except the TU datasets, all models operate under a supervised learning paradigm, where the results on validation sets are used to fine-tune hyperparameters. For MUTAG, PROTEINS, and ENZYMES, we use the 5-fold cross validation strategy described in (Errica et al., 2020).

Hyperparameters. The hyperparameters are selected via a Bayesian search strategy using the hyperparameters described in Table 5. For the TU datasets, the choice of hyperparameters is motivated by (Errica et al., 2020) and for the other datasets it is motivated by (Rampášek et al., 2022; Luo et al., 2025). Besides, hyperparameters described in Table 5, for the models that require attention heads (GraphGPS and GRIT), the number of attention heads is chosen between 1 head and 4 heads.

Table 5: Summary of the hyperparameters grid used for the experiments. Double square brackets $[\![]\!]$ denote integer valued interval, square brackets [] float valued interval and brackets $\{\}$ correspond to sets of values.

Datasets	Number of layers	Hidden dimension	Dropout	Learning rate	Batch Size	Epochs
MUTAG	$[\![2, 5]\!]$	$[\![32, 64]\!]$	[0, 0.5]	[0.001, 0.01]	16	500
ENZYMES	$[\![2, 5]\!]$	$[\![32, 64]\!]$	[0, 0.5]	0.01	{32, 128}	1000
PROTEINS	$[\![2, 5]\!]$	$[\![32, 64]\!]$	[0, 0.5]	0.01	{32, 128}	1000
OGBG-MolHIV	{3, 4, 5, 10}	$[\![64, 512]\!]$	[0.05, 0.1]	0.0001	32	150
Peptides-func	$[\![3, 5]\!]$	$[\![64, 256]\!]$	[0.05, 0.2]	0.001	$[\![16, 128]\!]$	300
Peptides-struct	$[\![3, 5]\!]$	$[\![64, 256]\!]$	[0.05, 0.2]	0.001	32	300

G.4 Ablation Study

We present results using alternative DAG architectures in Figure 7 and conclude that our DAG layer is a suitable model choice.