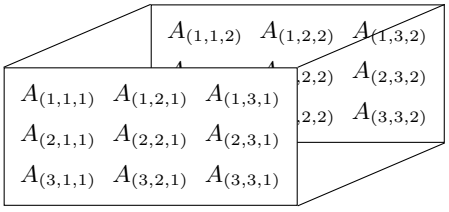
Supplementary Material of Relational Pooling

A. Tensor Representation and vec Operation on Graphs

We briefly provide a concrete example of the representation of graphs and the operation $\mathrm{vec}(G)$. Consider a graph with three vertices, one edge attribute at each edge, and two vertex attributes at each vertex. The connectivity structure and edge attributes are represented by the $3 \times 3 \times 2$ adjacency tensor $\mathbf{A}$ where $A_{(i,j,1)}$ denotes the value of the graph's adjacency matrix and $A_{(i,j,2)}$ denotes the value of the additional edge attribute, $i, j \in V = \{1, 2, 3\}$.



Observe that the possibility of attributed self-loops is contemplated but this representation is applicable both to graphs that have self-loops and those that do not. The vertex attributes are represented in a matrix

$$\boldsymbol{X}^{(v)} = \begin{pmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \\ X_{3,1} & X_{3,2} \end{pmatrix}.$$

A simple vec operation is shown below. The modeler is free to make modifications such as applying an MLP to the vertex attributes before concatenating with the edge attributes. Representing G by $\mathbf{A}$ and $\boldsymbol{X}^{(v)}$,

$$\begin{aligned}\mathrm{vec}(G) = \big(&A_{(1,1,1)}, A_{(1,1,2)}, A_{(1,2,1)}, A_{(1,2,2)}, \\ &A_{(1,3,1)}, A_{(1,3,2)}, X_{1,1}, X_{1,2}, A_{(2,1,1)}, A_{(2,1,2)}, \\ &A_{(2,2,1)}, A_{(2,2,2)}, A_{(2,3,1)}, A_{(2,3,2)}, X_{2,1}, X_{2,2}, \\ &A_{(3,1,1)}, A_{(3,1,2)}, A_{(3,2,1)}, A_{(3,2,2)}, A_{(3,3,1)}, \\ &A_{(3,3,2)}, X_{3,1}, X_{3,2}\big).\end{aligned}$$

Starting with the first vertex, each edge attribute (including the edge indicator) is listed, then the vertex attributes are added before doing the same with subsequent vertices. The vectorization method for k-ary type models is similar, except that we apply vec on induced subgraphs of size k.

B. More on Permutation-Invariance, WL-GNN Models, and Unique Identifiers

Here we elaborate on the addition of unique identifiers to graphs and implications for WL-GNN models. For simplicity, we consider undirected graphs with vertex attributes but no edge attributes, allowing us to simplify our notation to an adjacency matrix $\boldsymbol{A}$ and vertex attribute matrix $\boldsymbol{X}$. We also consider an oversimplified model with just one GNN layer ($L = 1$), the following aggregation scheme

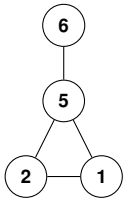
$$\mathbf{h}_u = \mathbf{x}_u + \sum_{v \in \mathcal{N}(u)} \mathbf{x}_v, \quad \forall u \in V,$$

and the following read-out function to yield a graph representation

$$\mathbf{h}_G = \sum_{v \in V} \mathbf{h}_v.$$

This can be expressed as $\mathbf{h}_G = \mathbf{1}^T(\boldsymbol{A} + I_{|V|})\boldsymbol{X}$ for adjacency matrix $\boldsymbol{A}$, vertex attribute matrix $\boldsymbol{X}$, identity matrix $I_{|V|}$, and where $\mathbf{1}^T$ is a row vector of ones.

For instance, we may observe the following graph with endowed vertex attributes. The numbers indicate vertex features, not labels.



We can represent this with an adjacency matrix and vertex attribute matrix as

$$\boldsymbol{A} = \begin{pmatrix} 0&0&0&1 \\ 0&0&1&1 \\ 0&1&0&1 \\ 1&1&1&0 \end{pmatrix}, \boldsymbol{X} = \begin{pmatrix} 6 \\ 2 \\ 1 \\ 5 \end{pmatrix}.$$

Here, $\mathbf{1}^T(\boldsymbol{A} + I_{|V|})\boldsymbol{X} = 41$. Equivalently, we might have chosen to represent this graph as

$$\boldsymbol{A}_{\pi,\pi} = \begin{pmatrix} 0&0&1&0 \\ 0&0&1&1 \\ 1&1&0&1 \\ 0&1&1&0 \end{pmatrix}, \boldsymbol{X}_{\pi} = \begin{pmatrix} 6 \\ 2 \\ 5 \\ 1 \end{pmatrix}$$

Here we swapped the third and fourth column of $\boldsymbol{X}$ and the third and fourth row and column of $\boldsymbol{A}$. Yet again, $\mathbf{1}^T(\boldsymbol{A}_{\pi,\pi} + I_{|V|})\boldsymbol{X}_{\pi} = 41$, as desired for isomorphic-invariant functions. We have chosen to assign scalar vertex attributes, but the invariance to permutation holds for vector vertex attributes.

Now, we propose assigning unique one-hot IDs after constructing the adjacency matrix, which corresponds to the following representations

$$\boldsymbol{A} = \begin{pmatrix} 0&0&0&1 \\ 0&0&1&1 \\ 0&1&0&1 \\ 1&1&1&0 \end{pmatrix}, \big[\boldsymbol{X} \bowtie I_{|V|}\big] = \begin{pmatrix} 6&1&0&0&0 \\ 2&0&1&0&0 \\ 1&0&0&1&0 \\ 5&0&0&0&1 \end{pmatrix}$$

$$\boldsymbol{A}_{\pi,\pi} = \begin{pmatrix} 0&0&1&0 \\ 0&0&1&1 \\ 1&1&0&1 \\ 0&1&1&0 \end{pmatrix}, \big[\boldsymbol{X}_{\pi} \bowtie I_{|V|}\big] = \begin{pmatrix} 6&1&0&0&0 \\ 2&0&1&0&0 \\ 5&0&0&1&0 \\ 1&0&0&0&1 \end{pmatrix}$$

(recall that $[\cdot \bowtie \cdot]$ denotes concatenation). Note that we effectively assign identifiers after constructing $\boldsymbol{A}$ and $\boldsymbol{X}$ (and similarly for $\boldsymbol{A}_{\pi,\pi}$, $\boldsymbol{X}_\pi$), so that the latter four columns of $\left[\boldsymbol{X} \bowtie I_{|V|}\right]$ and $\left[\boldsymbol{X}_\pi \bowtie I_{|V|}\right]$ are the same.

Now, $\mathbf{1}^T(\boldsymbol{A} + I_{|V|})\left[\boldsymbol{X} \bowtie I_{|V|}\right] = (41, 2, 3, 3, 4)$ yet $\mathbf{1}^T(\boldsymbol{A}_{\pi,\pi} + I_{|V|})\left[\boldsymbol{X}_\pi \bowtie I_{|V|}\right] = (41, 2, 3, 4, 3)$. This permutation sensitivity in the presence of unique IDs holds for more general WL-GNNs and not just the one considered here. Often $\mathbf{h}_G$ is fed forward through a linear or more complex layer to obtain the final graph-level prediction and this layer is usually permutation sensitive. Thus, we apply RP to GNNs with unique IDs to guarantee permutation invariance; meanwhile, the intuition for using unique IDs is to better distinguish vertices and thus create a more powerful representation for the graph.

B.1. An alternative approach to RP-GNN models

Next we present an equivalent but alternative representation of RP-GNN models (Equation 5) that may be simpler to implement in practice and provide an example. In the previous section, we described permuting the adjacency tensor and matrix of endowed vertex attributes, leaving the matrix of identifiers unchanged. Alternatively, with $\vec{f}$ modeled as an isomorphic-invariant Graph Neural Network, one may leave the former two unchanged and instead permute the matrix of identifiers. Thus, the alternative model becomes

$$\overline{\vec{f}}(G) = \frac{1}{|V|!} \sum_{\pi \in \Pi_{|V|}} \vec{f}\left(\mathbf{A}, \left[\boldsymbol{X}^{(v)} \bowtie (I_{|V|})_\pi\right]\right), \quad (11)$$

where $(I_{|V|})_\pi$ denotes a permutation of the rows of the identity matrix. The more tractable version discussed previously of assigning a one-hot encoding of the id $i \bmod m$ to node $i \in V$, for some $m \in \{1, 2, \ldots, |V|\}$ is still applicable. In this case, we replace $(I_{|V|})_\pi$ with a $|V| \times m$ matrix of m-bit one-hot identifiers, appropriately permuted by π.

For example, consider again the graph defined by adjacency matrix $\boldsymbol{A}$ and vertex features $\boldsymbol{X}$ given above. To evaluate $\vec{f}(\mathbf{A}_{\pi,\pi}, \boldsymbol{X}_\pi)$ when the permutation is given by $\pi(1, 2, 3, 4) = (2, 1, 3, 4)$, we could forward

$$A = \begin{pmatrix} 0&0&0&1 \\ 0&0&1&1 \\ 0&1&0&1 \\ 1&1&1&0 \end{pmatrix}, \left[\boldsymbol{X} \bowtie (I_{|V|})_\pi\right] = \begin{pmatrix} 6&0&1&0&0 \\ 2&1&0&0&0 \\ 1&0&0&1&0 \\ 5&0&0&0&1 \end{pmatrix}$$

through our model of $\vec{f}$. Previously the first row of $\left[\boldsymbol{X} \bowtie I_{|V|}\right]$ was $(6, 1, 0, 0, 0)$ whereas after permutation by π it became $(6, 0, 1, 0, 0)$, and so on. Both formulations discussed in this section were used in our experiments.

Proof of Theorem 2.1

Proof. Let Ω be a finite set of graphs $G = (V, E, \boldsymbol{X}^{(v)}, \mathbf{X}^{(e)}) = (\mathbf{A}, \boldsymbol{X}^{(v)})$ that includes all graph topologies for any given (arbitrarily large but finite) graph order, as well as the associated vertex and edge attributes from a finite set. Note that isomorphic graphs G and $G_{\pi,\pi}$ are considered distinct elements in Ω ($G_{\pi,\pi}$ denotes a permutation of $\mathbf{A}$ and $\boldsymbol{X}^{(v)}$). If $G = (\mathbf{A}, \boldsymbol{X}^{(v)})$, let $\mathbb{G}(G) = \{(\mathbf{A}_{\pi,\pi}, \boldsymbol{X}_\pi^{(v)}) : \pi \in \Pi_{|V|}\}$ denote the set of graphs that are isomorphic to G and have the same vertex and edge attribute matrices up to permutation. Consider a classification/prediction task where $G \in \Omega$ is assigned a target value $t(G)$ from a collection of $|\Omega|$ possible values, such that $t(G) = t(G')$ iff $G' \in \mathbb{G}(G)$. Clearly, this is the most general classification task. Moreover, by replacing the target value $t(G)$ with a probability $p(G)$ (measure), the above task also encompasses generative tasks over Ω. All we need to show is that $\overline{\vec{f}}$ of Equation 1 is sufficiently expressive for the above task.

We now consider a permutation-sensitive function $\vec{f}$ that assigns a distinct one-hot encoding to each distinct input graph $G = (\mathbf{A}, \boldsymbol{X}^{(v)})$. This $\vec{f}$ can be approximated arbitrarily well by a sufficiently expressive neural network (operating on the vector representation of the input) as these are known to be universal approximators (Hornik et al., 1989). Now, letting $G' \in \Omega$ be arbitrary, for all $G \in \mathbb{G}(G')$, we have

$$\begin{aligned} \overline{\vec{f}}(G) &= \frac{1}{|V|!} \sum_{\pi \in \Pi_{|V|}} \vec{f}(\mathbf{A}_{\pi,\pi}, \boldsymbol{X}_\pi^{(v)}) \\ &= \frac{1}{|V|!} \sum_{(\mathbf{A}', \boldsymbol{X}'^{(v)}) \in \mathbb{G}(G)} \vec{f}(\mathbf{A}', \boldsymbol{X}'^{(v)}) \\ &= \frac{1}{|V|!} \sum_{(\mathbf{A}', \boldsymbol{X}'^{(v)}) \in \mathbb{G}(G')} \vec{f}(\mathbf{A}', \boldsymbol{X}'^{(v)}) \\ &= \frac{1}{|V|!} \overline{\vec{f}}(G'), \end{aligned}$$

thus $\overline{\vec{f}}(G')$ is the unique fingerprint of the set $\mathbb{G}(G')$. Then, all we need is a function $\rho(\cdot)$ that takes the representation $\overline{\vec{f}}(G')$ and assigns the unique target value $t(G')$, satisfying the desired condition and proving that RP has maximal representation power over Ω. □

Proof of Theorem 2.2

Preliminaries. For an $n \times d_B$ matrix B and an $n \times d_C$ matrix C, write $D = [B \bowtie C]$ to denote their concatenation to form an $n \times (d_B + d_C)$ matrix D. Recall that RP-GNN adds a one-hot encoding of the *node id* to the node features. This *node id* is defined as its position in the adjacency matrix or tensor $\mathbf{A}_{\pi,\pi}$ for a permutation π. Let $I_{|V|}$ be a $|V| \times |V|$ identity matrix representing the one-hot encoding vectors

of node IDs 1 to $|V|$. We let α denote a maximally powerful WL-GNN, that is, a deep-enough WL-GNN satisfying the conditions of Theorem 3 in Xu et al. (2019). That is, the multiset functions for vertex aggregation and the graph-level readout are both injective over discrete node attributes. In accordance with Xu et al. (2019), we focus on graphs whose features live in a countable space. Finally, we denote multisets by $\{\{\ldots\}\}$.

Proof. We need to show that RP-GNN is strictly more expressive than any WL-GNN. More specifically, we will show that RP-GNN (1) maps isomorphic graphs to the same graph embedding, (2) maps nonisomorphic graphs to distinct embeddings whenever a WL-GNN does, and (3) can map pairs of nonisomorphic graphs to distinct graph embeddings even when a most-powerful WL-GNN maps them to the same embedding. Again, in the context of the proof, when we say two graphs are 'isomorphic' it is understood that they have the same topology and the same vertex/edge features up to permutation.We suppose that all pairs of graphs have the same number of vertices, denoted by n; if they have different numbers of vertices it is trivial to show that both RP-GNN and WL-GNN can represent them differently.

(1) Assume $G_1 = (\mathbf{A}_1, \boldsymbol{X}_1^{(v)})$ and $G_2 = (\mathbf{A}_2, \boldsymbol{X}_2^{(v)})$ are isomorphic graphs with the same features (up to permutation). Let $\left[(\boldsymbol{X}_j^{(v)})_\pi \bowtie I_n\right]$ be the new (RP-GNN) features for some permutation $\pi \in \Pi_n$ of graph G_j for $j \in \{1, 2\}$. Since G_1 is isomorphic to G_2, there exists a $\pi' \in \Pi_n$ such that $\mathbf{A}_1 = (\mathbf{A}_2)_{\pi',\pi'}$ and $\left[\boldsymbol{X}_1^{(v)} \bowtie I_n\right] = \left[(\boldsymbol{X}_2^{(v)})_{\pi'} \bowtie I_n\right]$. Thus

$$\begin{aligned}
\overline{\overline{f}}(G_1) &= \frac{1}{n!}\sum_{\pi\in\Pi_n} \alpha\Big((\mathbf{A}_1)_{\pi,\pi}, \Big[(\boldsymbol{X}_1^{(v)})_\pi \bowtie I_n\Big]\Big) \\
&= \frac{1}{n!}\sum_{\pi''\in\Pi_n''} \alpha\Big((\mathbf{A}_2)_{\pi'',\pi''}, \Big[(\boldsymbol{X}_2^{(v)})_{\pi''} \bowtie I_n\Big]\Big) \\
&= \overline{\overline{f}}(G_2),
\end{aligned}$$

where we define $\Pi_n'' = \{\pi \circ \pi' : \pi \in \Pi_n\}$ and observe that $\Pi_n'' = \Pi_n$. Thus, no pairs of isomorphic graphs will be mapped to different representations by an RP-GNN $\overline{\overline{f}}$.

(2) Assume now that $G_1 = (\mathbf{A}_1, \boldsymbol{X}_1^{(v)})$ and $G_2 = (\mathbf{A}_2, \boldsymbol{X}_2^{(v)})$ are nonisomorphic graphs with discrete attributes successfully deemed nonisomorphic by the WL test. Theorem 3 of Xu et al. (2019) implies

$$\alpha(\mathbf{A}_1, \boldsymbol{X}_1^{(v)}) \neq \alpha(\mathbf{A}_2, \boldsymbol{X}_2^{(v)}).$$

Next, we can always construct an α' which has the same weights for the affine transformation over the endowed attributes as α but zero weights for the affine transformation over the RP-specific identifiers I_n such that

$$\alpha'\Big(\mathbf{A}_{\pi,\pi}, \Big[(\boldsymbol{X}^{(v)})_\pi \bowtie I_n\Big]\Big) = \alpha\Big(\mathbf{A}_{\pi,\pi}, \boldsymbol{X}_\pi^{(v)}\Big)$$

for any $G = (\mathbf{A}, \boldsymbol{X}^{(v)})$. Note that α' is isomorphic-invariant since α is by construction; indeed, α' ignores its permutation-sensitive part. Thus,

$$\begin{aligned}
\overline{\overline{f}}(G_1) &= \frac{1}{n!}\sum_{\pi\in\Pi_n} \alpha'\Big((\mathbf{A}_1)_{\pi,\pi}, \Big[(\boldsymbol{X}_1^{(v)})_\pi \bowtie I_n\Big]\Big) \\
&= \alpha\Big(\mathbf{A}_1, \boldsymbol{X}_1^{(v)}\Big) \\
&\neq \alpha\Big(\mathbf{A}_2, \boldsymbol{X}_2^{(v)}\Big) \\
&= \frac{1}{n!}\sum_{\pi\in\Pi_n} \alpha'\Big((\mathbf{A}_2)_{\pi,\pi}, \Big[(\boldsymbol{X}_2^{(v)})_\pi \bowtie I_n\Big]\Big) \\
&= \overline{\overline{f}}(G_2).
\end{aligned}$$

Therefore, RP-GNN can map graphs that WL-GNNs can distinguish to different representations, completing our proof of part (2).

(3) Finally, we construct an example to show that RP-GNN is more expressive than WL-GNN. Consider the circulant graphs with different skip links in Figure 1. We show that these two (pairwise nonisomorphic) graphs can have different representations by RP-GNN but cannot be represented as distinct by WL-GNN. Let $G_1 = (\mathbf{A}_1, \boldsymbol{X}^{(v)})$ denote the graph $\mathcal{G}_{\text{skip}}(M = 11, R = 2)$ and $G_2 = (\mathbf{A}_2, \boldsymbol{X}^{(v)})$ denote the graph $\mathcal{G}_{\text{skip}}(M = 11, R = 3)$, where $\boldsymbol{X}^{(v)} = c\mathbf{1}$, a vector of $c \in \mathbb{R}$. It is not hard to show that WL-GNN cannot give different representations to G_1 and G_2, as the WL test fails in these graphs (Arvind et al., 2017; Cai et al., 1992; Fürer, 2017) and the most powerful WL-GNN is just as powerful as the WL test (Xu et al., 2019).

To show that RP-GNN is capable of giving different representations, we first show that for any given permutation π of G_1, there is no permutation π' of G_2 such that $\alpha((\mathbf{A}_1)_{\pi,\pi}, \left[(\boldsymbol{X}_1^{(v)})_\pi \bowtie I_n\right]) = \alpha((\mathbf{A}_2)_{\pi',\pi'}, \left[(\boldsymbol{X}_2^{(v)})_{\pi'} \bowtie I_n\right])$ (note that α is a most-powerful GNN and thus not a constant function). In this part of the proof, for simplicity and without loss of generality, consider π a permutation such that the vertices are numbered sequentially from $1, 2, \ldots, n$ clockwise around the circle in Figure 1. Then, node 3 in G_1 has neighbors $N_3 = \{1, 2, 4, 5\}$ and node 4 has neighbors $N_4 = \{2, 3, 5, 6\}$, with intersection $N_3 \cap N_4 = \{2, 5\}$. However, in G_2, no two nodes share two neighbors. Therefore, the multisets of all neighborhood attribute sequences for both permuted graphs – denoted in $(G_l)_{\pi,\pi}$ as $\left\{\left\{(\mathbf{h}_{l,u}^{(0)})_{u\in N_v}\right\}\right\}_{v\in V_l}$, for $l \in \{1, 2\}$, where the $\mathbf{h}^{(0)}$ terms include the rows of I_n –

will be distinct. Thus a most powerful α with the recursion in Equation 4 will map them to distinct collections of vertex embeddings. Thus, the graph embeddings $\mathbf{h}_{G_l}$, obtained by applying an injective read-out function to $\left\{\left\{\mathbf{h}_{l,v}^{(1)}\right\}\right\}_{v\in V_l}$ will be distinct: $\mathbf{h}_{(G_1)_{\pi,\pi}} \neq \mathbf{h}_{(G_2)_{\pi',\pi'}}$, as desired.

As no representation of G_2 can match any representation of G_1, we can find a function $g(\cdot)$ that, when composed with α, ensures that the sum in Equation 1 gives different values for G_1 and G_2 by Lemma 5 of Xu et al. (2019) (or Theorem 2 of Zaheer et al. (2017)). Since we can always redefine $\alpha' = g \circ \alpha$ and α' is still a WL-GNN, we conclude our proof. □

Proposition 2.1

The following proposition regarding the convergence of π-SGD was stated in the paper:

Proposition 2.1. *π-SGD stochastic optimization enjoys properties of almost sure convergence to optimal $\boldsymbol{W}$ under conditions similar to SGD (listed in Supplementary).*

Here we list the relevant conditions. Murphy et al. (2019) point out that π-SGD can be characterized by the work of Younes (1999); Yuille (2005) and is a a familiar application of stochastic approximation algorithms already used in training neural networks.

In particular, the following assumptions are made:

1. There exists a constant $M > 0$ such that for all $\boldsymbol{W}$, $-\boldsymbol{G}_t^T \boldsymbol{W} \leq M\|\boldsymbol{W} - \boldsymbol{W}^\star\|_2^2$, where $\boldsymbol{G}_t$ is the true gradient for the full batch over all permutations and $\boldsymbol{W}^\star$ is an optimum.
2. there exists a constant $\delta > 0$ such that for all $\boldsymbol{W}$, $E_t[\|\mathbf{Z}_t\|_2^2] \leq \delta^2(1 + \|\boldsymbol{W}_t - \boldsymbol{W}_t^\star\|_2^2)$, where $\mathbf{Z}_t$ is the random gradient of the loss w.r.t. weights at step t and the expectation is taken with respect to all the data prior to step t.

If these assumptions are satisfied, then π-SGD (as with SGD) converges to a fixed point with probability one.

Proof of Proposition 2.2

We restate the proposition for completeness.

Proposition. *The RP in Equation 10 requires summing over all k-node induced subgraphs of G, thus saving computation when $k < |V|$, reducing the number of terms in the sum from $|V|!$ to $\frac{|V|!}{(|V|-k)!}$.*

Proof. k-ary RP needs to iterate over the k-node induced subgraphs of G ($\binom{|V|}{k}$ subgraphs), but for each subgraph there are $k!$ different ways to order its nodes, resulting in $\frac{|V|!}{(|V|-k)!}$ evaluations of $\vec{f}$. □

Proof of Proposition 2.3

We restate the proposition for completeness.

Proposition. *$\overline{\overline{f}}^{(k)}$ becomes strictly more expressive as k increases. That is, for any $k \in \mathbb{N}$, define $\mathcal{F}_k$ as the set of all permutation-invariant graph functions that can be represented by RP with k-ary dependencies. Then, $\mathcal{F}_{k-1}$ is a* proper *subset of $\mathcal{F}_k$. Thus, RP with k-ary dependencies can express any RP function with $(k-1)$-ary dependencies, but the converse does not hold.*

Proof. ($\mathcal{F}_{k-1} \subset \mathcal{F}_k$): Consider an arbitrary element $\overline{\overline{f}}^{(k-1)} \in \mathcal{F}_{k-1}$, and write $\vec{f}(\mathbf{A}[1:(k-1),1:(k-1),:], \boldsymbol{X}^{(v)}[1:(k-1),:]; \boldsymbol{W})$ for its associated permutation-sensitive RP function. Also consider $\overline{\overline{f}}^{(k)} \in \mathcal{F}_k$ and let $\vec{f}'$ be its associated permutation-sensitive RP function. For any tensor $\mathbf{A}$ and attribute matrix $\boldsymbol{X}^{(v)}$, we can define $\vec{f}'(\mathbf{A}[1:k,1:k,:], \boldsymbol{X}^{(v)}[1:k,:]; \boldsymbol{W}) = \vec{f}(\mathbf{A}[1:(k-1),1:(k-1),:], \boldsymbol{X}^{(v)}[1:(k-1),:]; \boldsymbol{W})$. Thus, $\overline{\overline{f}}^{(k-1)} \in \mathcal{F}_k$ and because $\overline{\overline{f}}^{(k-1)}$ is arbitrary, we conclude $\mathcal{F}_{k-1} \subset \mathcal{F}_k$.

($\mathcal{F}_k \not\subset \mathcal{F}_{k-1}$): The case where $k = 1$ is trivial, so assume $k > 1$. We will demonstrate $\exists \overline{\overline{f}}^{(k)} \in \mathcal{F}_k$ such that $\overline{\overline{f}}^{(k)} \notin \mathcal{F}_{k-1}$. Let $\overline{\overline{f}}^{(k)}$ and $\overline{\overline{f}}^{(k-1)}$ be associated with $\vec{f}^{(k)}$ and $\vec{f}^{(k-1)}$, respectively.

Task. Consider the task of representing the class of circle graphs with skip links shown in Figure 1. Let $G_k \in \mathcal{G}_{\text{skip}}(M_k, k)$ and $G_{k+1} \in \mathcal{G}_{\text{skip}}(M_k, k+1)$ where M_k is any prime number satisfying $M_k > 2(k-1)(k+1)$. That is, G_k and G_{k+1} are circulant skip length graphs with the same number of vertices and skip lengths of k and $k+1$, respectively. Note that $M_k > k+1$ is prime and thus it is co-prime with both k and $k+1$; further, $M_k - 1 > k+1$ so the conditions for creating the CSL graph in Definition 2.1 are indeed satisfied. To complete the proof, we need to show that (1) there is a $\overline{\overline{f}}^{(k)}$ capable of distinguishing G_k from G_{k+1} but (2) no such $\overline{\overline{f}}^{(k-1)}$ exists.

Denote $G_R = (\boldsymbol{A}_R, \boldsymbol{X}^{(v)})$, $R \in \{k, k+1\}$, where $\boldsymbol{X}^{(v)} = c\mathbf{1}$ for some $c \in \mathbb{R}$ for both graphs, as there are no vertex features, and where $\boldsymbol{A}_R$ represents an adjacency matrix for G_R (there are no edge features).

(1) A k-ary $\overline{\overline{f}}^{(k)}$ that can distinguish between G_k and G_{k+1}. We will define $\vec{f}^{(k)}$ in terms of a composition with a canonical orientation. In particular, we only allow the orientation of $\boldsymbol{A}$ (and $\boldsymbol{X}^{(v)}$) that arises from first generating an edgelist by the scheme described in Definition 2.1 and

then constructing the adjacency matrix from it in the usual way. For either graph, we define $\vec{f}^{(k)}((\boldsymbol{A}_R)_{\pi,\pi}[1:k,1:k,:], \boldsymbol{X}_\pi^{(v)}[1:k,:]) = 0$ for all permutations that do not yield this 'canonical' adjacency matrix.

Under this orientation, the $k \times k$ submatrix $\boldsymbol{A}_k[1:k,1:k]$ of G_k will have more nonzero elements than that of G_{k+1}, $\boldsymbol{A}_{k+1}[1:k,1:k]$. The relevant induced subgraph of size k in G_k will include a pair of vertices that are k 'hops' away which will thus be connected by an edge, whereas in G_{k+1}, the skip length is too long so its induced subgraphs of size k will have fewer edges. Therefore, it suffices to let $\vec{f}^{(k)}$ count the number of nonzero elements in the (properly oriented) submatrices presented to it.

(2) No $(k-1)$-ary $\vec{f}^{(k-1)}$ can distinguish between G_k and G_{k+1}. We will show that the induced subgraphs of size $k-1$ are "the same" in both G_k and G_{k+1}, which will imply that no satisfactory $\vec{f}^{(k-1)}$ can be constructed. In particular, if we denote by $\mathscr{L}_{k-1}(G_k)$ the multiset of induced subgraphs of size $k-1$ in G_k and by $\mathscr{L}_{k-1}(G_{k+1})$ the multiset of induced subgraphs of size $k-1$ in G_{k+1}, it can be shown that $\mathscr{L}_{k-1}(G_k)$ and $\mathscr{L}_{k-1}(G_{k+1})$ are equivalent in the following sense. There exists a bijection ϕ between these finite multisets such that for every $H \in \mathscr{L}_{k-1}(G_k)$, $\phi(H) \in \mathscr{L}_{k-1}(G_{k+1})$ is isomorphic to H. For example, the multisets (with arbitrarily labeled vertices) $\{\{$ ①-②-③ , ②-③-④ , ⑤-⑥-⑦ $\}\}$ and $\{\{$ ①-③-⑤ , ②-⑤-⑦ , ⑤-⑦-⑨ $\}\}$ have an isomorphism-preserving bijection between them and will thus be considered equal.

In the interest of brevity, what follows is a sketch. We first observe that we only need to consider the multisets of induced subgraphs that include vertex $0 \in V = \{0,1,\ldots,M_k-1\}$ due to the vertex transitivity of the CSL graphs. Next, we observe that we only need to consider the multisets of 'maximally connected' induced subgraphs of size $k-1$. By 'maximally connected', we mean an induced subgraph of size $k-1$ such that no more edges from G_R, $R \in \{k, k+1\}$, can be added without adding a k^{th} vertex to the induced subgraph. Indeed, once we show that a bijection exists between maximally connected induced subgraphs of size $k-1$, it follows that such a bijection exists for *any* connected induced subgraphs of size $k-1$ since these can be formed by deleting any edge that does not render the induced subgraph disconnected. Then, viewing disconnected graphs as the disjoint union of connected components, a similar argument to the one applied for connected induced subgraphs can be used to complete the argument for any possible induced subgraph of size $k-1$.

We can construct all such maximally connected subgraphs including $0 \in V$ in both G_R for $R \in \{k, k+1\}$ by forming recursive sequences on the integers $\{0,1,\ldots,M_k-1\}$ with addition mod M_k (see for instance Definition 2.1); the key difference in these sequences is whether $R = k$ or $R = k+1$ can be added to or subtracted from the previous value in the sequence. We will call the former a k-sequence and the latter a $(k+1)$-sequence. In either case, distinct sequences may result in the same induced subgraphs but we can simply take one representative from each equivalence class.

Importantly, these sequences can be constructed in a way that abstracts from either underlying graph G_k or G_{k+1}. Due to our choice of M_k, the recursive sequences never 'wrap around' the graph and can be informally thought of as a recursive sequence on the integers of a bounded interval with 0 in the middle rather than a circle with skip links. In particular, we can define recursive sequences on the set of integers $\{-(k+1)(k-1),\ldots,(k+1)(k-1)\}$ with regular addition to construct the same induced subgraphs (-1 corresponds to vertex $(M_k-1) \in V$ and so on, in either case). Then it becomes clear that there is an isomorphism-preserving bijection between the induced subgraphs formed by recursive $(k+1)$-sequences and k-sequences on this bounded interval of integers; any sequence defined in terms of adding or subtracting $k+1$ can be replaced by one that adds or subtracts k (and vice versa), which completes the proof.

□

C. Further Details of the Experiments

C.1. Relational Pooling and Graph Structure Representation on CSL Graphs

Our GIN architecture uses five layers of recursion, where every $\text{MLP}^{(l)}$ has two hidden layers with 16 neurons in the hidden layers. The graph embedding is mapped to the output through a final linear layer $\text{softmax}(\mathbf{h}_G^T\mathbf{W})$. $\epsilon^{(l)}$ is treated as a learnable parameter. With standard GIN, since the vertex attributes are not one-hot encoded (they are constants), we first apply an MLP embedding before computing the first update recursion (as in Xu et al. (2019)). Since RP-GIN utilizes one-hot IDs, we do not need an MLP embedding in the first update. For these experiments, we assign one-hot encoding of $i \bmod 10$ for $i \in \{1,2\ldots,|V|=41\}$ – rather than completely unique IDs – which facilitates learning. We train with π-SGD, applying one random permutation of each adjacency matrix at each epoch. For inference, we average the score over five random permutations of each graph, as in Remark 2.1. Figure 3 shows the stronger performance of RP-GIN on this task.

Both models are trained for 1000 epochs using ADAM (Kingma & Ba, 2015) for optimization.

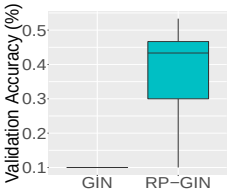


Figure 3: RP-GNN is more powerful than WL-GNN in a challenging 10-class classification task.

Table 3: Datasets used in our experiments.

Data Set	Number of Compounds	Number of Tasks
HIV	41,127	1
MUV	93,087	17
Tox 21	6,284	12

For the cross-validation, we use five random initializations at each fold. The folds are such that the classes are balanced in both training and validation.

Models are trained on CPUs but on machines with multiple CPUs; PyTorch inherently multithreads the execution.

C.2. Predicting Molecular Properties

Here we provide additional details on the molecular experiments. (1) For the models based on Graph Convolution (Duvenaud et al., 2015; Altae-Tran et al., 2017), we extend the architecture provided from DeepChem and the MoleculeNet project. Following them, the learning rate was set to 0.003, we trained with mini-batches of size 96, and used the Adagrad optimizer(Duchi et al., 2011). Models were trained for 100 epochs. Training was performed on 48 CPUs using the inherent multithreading of DeepChem.

Note that we re-trained DeepChem models using this fewer number of epochs to make results comparable. That being said, many models reached optimal performance before the last epoch; we use the model with best validation-set performance for test-set prediction.

(2) For the so-called RNN and CNN models, all MLPs have one hidden layer with 100 neurons. We used the Adam optimizer (Kingma & Ba, 2015), again training all models with mini-batches of size 96 and 50 epochs. We performed a hyperparameter line search over the learning rate, with values in $\{0.003, 0.001, 0.01, 0.03, 0.1, 0.3\}$. Training was performed on GeForce GTX 1080 Ti GPUs. To model the RNN, we use an LSTM with 100 neurons and use the long term memory as output.

RP-Duvenaud When we train RP-Duvenaud, we follow any training particulars as in the DeepChem implementation. For instance, DeepChem's implementation computes a weighted loss which penalizes misclassification differently depending on the task, and they compute an overall performance metric by taking the mean of the AUC across all tasks (see Table 3). One difference is that the DeepChem recommends either metrics PRC-AUC or ROC-AUC and splits "random" or "scaffold" depending on the dataset under consideration. Since ROC-AUC and random splits were the most commonly used among the three datasets we chose, we decided – before training any models – to use random splits and ROC-AUC for every dataset for simplicity. We also note that the authors of MoleculeNet report ROC-AUC scores on all three datasets. Regarding the sizes of the train/validation/test splits, we used the default values provided by DeepChem.

We implement the model that assigns unique IDs to atoms by first finding the molecule with the most atoms across training, validation, and test sets, and then appending a feature vector of that size to the endowed vertex attributes. That is, if the largest molecule has A atoms, we concatenate a vector of length A of one-hot IDs to the existing vertex attributes (for every vertex in each molecule).

CNNs and RNNs We explore $k = 20$-ary RP with $\vec{f}$ as a CNN, learned with π-SGD. At each forward step, we run a DFS from a different randomly-selected vertex to obtain a $20 \times 20 \times 14$ subtensor of $\mathbf{A}$ (there are 14 edge features), which we feed through two iterations of conv $\rightarrow$ ReLU $\rightarrow$ MaxPool to obtain a representation $\boldsymbol{h}_{\mathbf{A}}$ of $\mathbf{A}$. The corresponding vertex attributes are fed through an MLP and concatenated with $\boldsymbol{h}_{\mathbf{A}}$ to obtain a representation $\boldsymbol{h}_G$ of the graph which in turn is fed through an MLP to obtain the predicted class (see also Equation 3). Zero padding was used to account for the variable-sized molecules. Twenty initial vertices for the DFS (i.e. random permutations) were sampled at inference time. Table 2 shows that the CNN $\vec{f}$ underperforms in all tasks.

We also consider RP with an RNN as $\vec{f}$ learned with π-SGD, starting with a DFS to yield a $|V| \times |V| \times 14$ subtensor. For $\vec{f}$, we treat the edge features of a given vertex as a sequence: for vertex v, we apply an LSTM to the sequence $(\mathbf{A}_{v,1,\cdot}, \mathbf{A}_{v,2,\cdot}, \ldots, \mathbf{A}_{v,|V|,\cdot})$ and extract the long-term state. We also take the vertex attributes and pass them through an MLP. The long term state and output of the MLP are concatenated, ultimately forming a representation for every vertex (and its neighborhood) which we view as a second sequence. We apply a second LSTM and again extract the long term state, which can be denoted $\boldsymbol{h}_G$, the embedding of the graph. Last, $\boldsymbol{h}_G$ is forwarded through an MLP yielding a class prediction. Twenty starting vertices (i.e. permutations) were sampled at inference time. Variability was quantified with 5 random train/val/test splits for both neural network based models. Interestingly, Table 2 shows that the RP-RNN approach performs reasonably well in the Tox21 dataset, while underperforming in other datasets. Future

work is needed to determine those tasks for which the RNN and CNN approaches are better suited.

π-SGD To train with π-SGD, we sample a different random permutation of the graph at each forward pass. In the case of RP-Duvenaud, this involves assigning permutation-dependent unique IDs at each forward step (as in Equation 5). In our implementation, we achieve this by building a new DeepChem object for the molecule at each forward pass. This operation is expensive but we did not consider refined code optimizations for this work. In general, with properly optimized code, sampling permutations need not be as expensive and allows for a tractable and theoretically justified procedure. Looking ahead to the test data in order to find the largest molecule in test and validation corresponds to using domain knowledge and the modeling choice that the resulting model will only work on molecules with at most A atoms. It is not hard to construct a similar model that does not rely on this look-ahead mechanism, such as assigning a one-hot encoding of $i \bmod A_{\text{observed}}$ where $i \in \{1, 2, \ldots, |V|\}$ and A_{observed} is the largest molecule observed in the model building phase.

Molecule dataset details Details on the molecular datasets are shown in Table 3. The observant reader may notice that we report a different number of Tox21 molecules than in Wu et al. (2018). This resulted from simultaneously finding (1) that the validation and testing Python objects for Tox21 were empty when we first loaded them in the early stage of development and (2) comments in the DeepChem source code that led us to believe that this was expected behavior. We thus split up the 'training' dataset rather than using the provided splits. This treatment was the same for all models, making the comparison fair.

The number of molecules in each dataset with greater than $k = 10, 20, 30, 40, 50$ molecules are 98.18%, 63.71%, 22.30%, 7.71%, 3.59% for HIV; 99.93%, 75.33%, 12.30%, 0.03%, 0.00% for MUV; and 78.07%, 33.63%, 10.39%, 3.90%, 1.97% for Tox21.

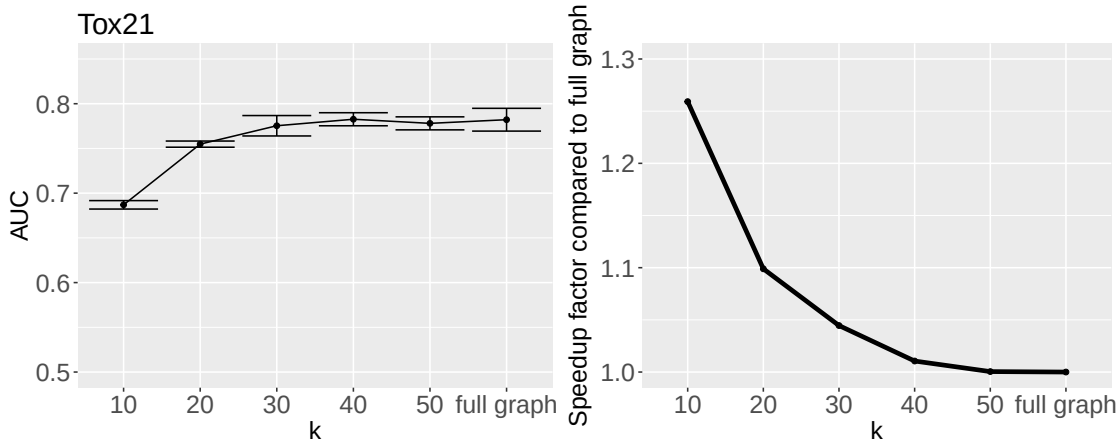


Figure 4: Training time and model performance of k-ary models for the Tox21 task. Test-set AUC was computed for five different random splits of train/validation/test: we show the mean $\pm$ one standard deviation. We also show the speedup factor for training: time to train on the full graph divided by time for k-ary model. Training was performed on 48 CPUs, making use of PyTorch's inherent multithreading.

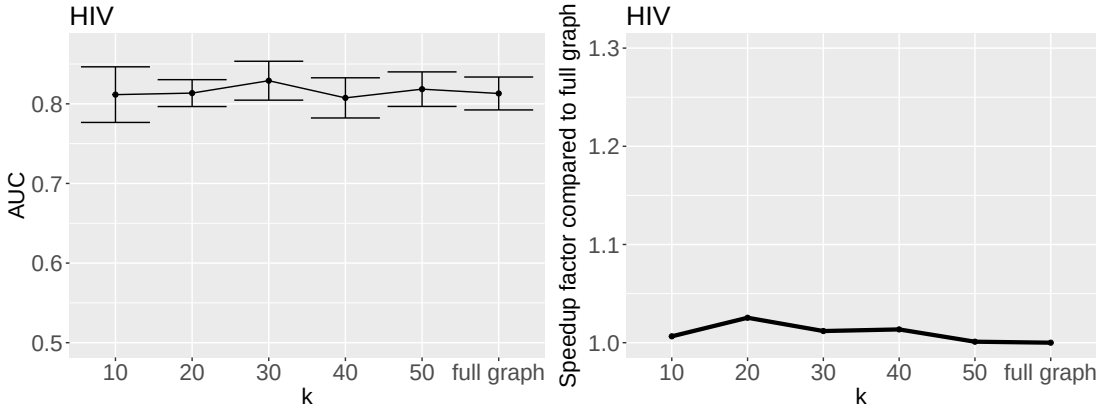


Figure 5: Training time and model performance of k-ary models for the HIV task.

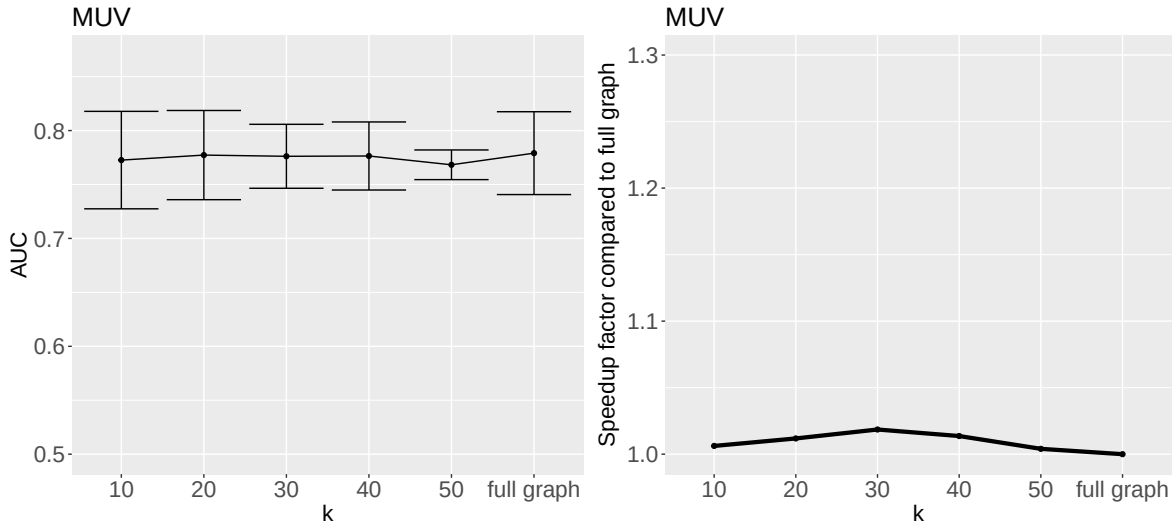


Figure 6: Training time and model performance of k-ary models for the MUV task.