Transformers Generalize DeepSets and Can be Extended to Graphs & Hypergraphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

We present a generalization of Transformers to any-order permutation invariant data (sets, graphs, and hypergraphs). We begin by observing that Transformers generalize DeepSets, or first-order (set-input) permutation invariant MLPs. Then, based on recently characterized higher-order invariant MLPs, we extend the concept of self-attention and propose order-k Transformers for higher-order data ($k = 2$ for graphs and $k > 2$ for hypergraphs). Unfortunately, order-k Transformers turn out to have prohibitive memory complexity $\mathcal{O}(n^{2k})$ given a hypergraph with n nodes. To address this problem, we present sparse order-k Transformers that have $\mathcal{O}(m^2)$ complexity for a given hypergraph with m hyperedges, and further adopt kernel attention approaches to reduce the complexity to $\mathcal{O}(m)$. In particular, we show that the sparse second-order Transformers with kernel attention are strictly more expressive than message passing operations while having an asymptotically identical complexity. Our models achieve significant performance improvement over invariant MLPs in graph classification and set-to-graph prediction tasks, and are competitive to state-of-the-art graph neural networks. [todo: elaborate.]

1 Introduction

Graph is a universal data modality used to model social networks [17], chemical compounds [6], biological structures [5], and interactions in particle physics [12]. Contemporary graph neural networks (GNNs) adopt a message passing scheme [25, 24, 20], where the node features are propagated and aggregated recurrently according to the neighborhood structure of a graph. Despite the simplicity, this recurrent nature makes them difficult to discover global dependency [7], being limited to small models due to vanishing gradient problem [2], and having a limited representation power [21]. Although recent works on message passing neural networks have achieved significant improvements via normalization [11], improved pooling [19, 21], and dilated convolution [14], the fundamental limitations of recurrent propagation still remains.

An alternative and more general direction here is to view graphs, more generally hypergraphs, as a type of data that is invariant to permutation of its nodes [15]. In this viewpoint, a (hyper-)graph data point G is represented as a tuple $(V, \mathbf{A})$, where V is a set of n nodes and tensor $\mathbf{A} \in \mathbb{R}^{n^k \times d}$ encodes d-dimensional feature vectors attached to its hyperedges[1]. The order k of the tensor $\mathbf{A}$ indicates the type of the encoded data. First-order tensor represents a *set* of node features (e.g., point cloud, bag-of-words of a sentence) where $\mathbf{A}_i$ is the feature of i-th node, and second-order tensor represents *edge* features where $\mathbf{A}_{i,j}$ is the feature of edge (i, j). Generally, an order-k tensor encodes *hyperedge* features where $\mathbf{A}_{i_1,...,i_k}$ is the feature of the hyperedge represented by $(i_1, ..., i_k)$. In this framework, we can represent a conventional graph with a node set V, node features $X \in \mathbb{R}^{n \times d}$, and adjacency matrix $A \in \mathbb{R}^{n \times n}$, as a tuple $G = (V, \mathbf{A})$ with the second-order $\mathbf{A} \in \mathbb{R}^{n \times n \times (1+d)}$

[1] A hyperedge is an ordered subset of the nodes V.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

constructed by placing the node features on top of the diagonals of the adjacency matrix. Now, our problem of interest is making neural networks for order-k tensors $\mathbf{A}$. A way to do so is to identify the full collection of *permutation equivariant* linear layers that respect the symmetry of $\mathbf{A}$ to node reordering, and composing an MLP as a stack of these layers with nonlinear activations [15]. In theory, such invariant MLPs should be more powerful than message-passing networks because they can consider various global interactions within a hypergraph, which could be difficult for message passing due to its locality. However, these networks are currently not widely adopted due to several major drawbacks, including worse performance compared to message passing networks and high asymptotic memory complexity.

Contributions In this work, we enhance the low performance and high asymptotic complexity of invariant MLPs by enhancing the internal computations, exploiting sparsity of data, and adopting computation tricks. Our contribution can be summarized as follows. First, based on an observation that Transformer encoder generalizes first-order equivariant linear layers, we extend the concept of self-attention to higher orders and propose higher-order Transformer layers that generalize higher-order linear layers. Second, While Transformer layers with input and output orders k, l has $\mathcal{O}(n^{k+l})$ asymptotic complexity, we show that by simply assuming sparse input and output we obtain $\mathcal{O}(e^2)$ complexity given input hypergraph with e hyperedges. Further adopting kernel attention approaches, we propose a variant that reduces the complexity to $\mathcal{O}(e)$ while being strictly more expressive than message passing neural networks. Finally, We test the higher-order Transformers on synthetic dataset, benchmark graph classification, and set2graph prediction tasks that involves third-order hypergraphs. Higher-order Transformers achieve significant improvements over invariant MLPs, and are highly competitive in performance and scalability to the state-of-the-art GNNs.

2 Preliminary

Let us define some notations. We denote a set by $\{a, b, ..., c\}$, an ordered set (tuple) by $(a, b, ..., c)$. We denote $[n] = \{1, 2, ..., n\}$ and use multi-index $\mathbf{i} \in [n]^k$ to denote a k-tuple of indices, $\mathbf{i} = (i_1, i_2, ..., i_k)$. Let S_n be the permutation group on a set with n elements (the set of all permutations of $[n]$). $\pi \in S_n$ acts on multi-index $\mathbf{i} \in [n]^k$ entrywise by $\pi(\mathbf{i}) = (\pi(i_1), \pi(i_2), ..., \pi(i_k))$, and acts on order-$k$ tensors $\mathbf{A} \in \mathbb{R}^{n^k \times d}$ by $(\pi \cdot \mathbf{A})_{\mathbf{i},j} = \mathbf{A}_{\pi^{-1}(\mathbf{i}),j}$ $(j \in [d])$.

Invariance and equivariance The problem of interest is parameterizing a functional relation $f(\mathbf{A}) \approx T$ between tensor $\mathbf{A}$ and target T in a way that respects symmetry of $\mathbf{A}$ to node permutation. If T is a single output vector, we ask that f is *permutation invariant*, satisfying $f(\pi \cdot \mathbf{A}) = f(\mathbf{A})$; If T is a tensor, $T = \mathbf{T}$, we ask that f is *permutation equivariant*, satisfying $f(\pi \cdot \mathbf{A}) = \pi \cdot f(\mathbf{A})$, for all $\pi \in S_n$ and $\mathbf{A} \in \mathbb{R}^{n^k \times d}$. Under the standard paradigm of neural networks where f is defined as an alternating stack of linear layers and non-linear activations, the problem of constructing f reduces to finding invariant and equivariant *linear* layers.

Invariant and equivariant linear layers In this paragraph, we summarize the work of Maron et al. [2019] that characterized the space of all invariant linear layers $L_{k \to 0} : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{d'}$ and equivariant linear layers $L_{k \to l} : \mathbb{R}^{n^k \times d'} \to \mathbb{R}^{n^l \times d'}$. Note that an equivariant layer's input and output can have different orders. e.g., $L_{2 \to 1}$ consumes edge features and outputs node features. An invariant layer is a special case of equivariant layer with $l = 0$. [todo: Illustration + intuitive explanation on global interaction modeling]

To write those layers, let us define an equivalence relation $\sim$ in the index space of tensors $\mathbb{R}^{n^k}$. For multi-indices $\mathbf{i}, \mathbf{j} \in [n]^k$, we set $\mathbf{i} \sim \mathbf{j}$ iff $\mathbf{i}$ and $\mathbf{j}$ have the same equality pattern, or $\mathbf{i}_a = \mathbf{j}_a \Leftrightarrow \mathbf{i}_b = \mathbf{j}_b$ for all $a, b \in [k]$. The equivalence relation partitions the index set $[n]^k$ into a set of equivalence classes, denoted as $[n]^k/_\sim$. Each equivalence class in $[n]^k/_\sim$ can be represented by a unique partition of $[k]$, where each set in the partition represents maximal set of identical values. For $k = 2$, we have $[n]^2/_\sim = \{\gamma_1, \gamma_2\}$ with $\gamma_1 = \{\{1, 2\}\}, \gamma_2 = \{\{1\}, \{2\}\}$, where γ_1 represents all multi-indices (i, j) s.t. $i = j$, and γ_2 represents all (i, j) s.t. $i \neq j$. As each γ is associated with a unique partition of $[k]$, we can write $[n]^k/_\sim = \{\gamma_1, ..., \gamma_{\mathrm{b}(k)}\}$ where $\mathrm{b}(k)$ is the k-th Bell number, or the number of possible partitions of a k-element set.

For each equivalence class $\gamma \in [n]^k/_\sim$, we define a corresponding *basis tensor* $\mathbf{B}^\gamma \in \mathbb{R}^{n^k}$ as follows:

$$\mathbf{B}^\gamma_{\mathbf{i}} = \begin{cases} 1 & \mathbf{i} \in \gamma \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

For example, for $k = 2$, $\mathbf{B}^{\gamma_1}$ is the identity matrix I_n and $\mathbf{B}^{\gamma_2}$ is $1_n 1_n^\top - I_n$ where $1_n \in \mathbb{R}^n$ is a vector of ones. Notably, it turns out that the collection of basis tensors $\{\mathbf{B}^\gamma | \gamma \in [n]^k/_\sim\}$ forms an orthogonal basis for the space of order-k tensors $\mathbf{L} \in \mathbb{R}^{n^k}$ invariant under node permutation ($\pi \cdot \mathbf{L} = \mathbf{L}$). Therefore, the basis tensors can be used to construct the space of linear layers $L_{k \to l}$. For $\mathbf{i} \in [n]^k$, $\mathbf{j} \in [n]^l$, $\lambda \in [n]^k/_\sim$, and $\mu \in [n]^{k+l}/_\sim$, we write two kinds of basis tensors for weights and biases respectively:

$$\mathbf{B}^\mu_{\mathbf{i},\mathbf{j}} = \begin{cases} 1 & (\mathbf{i},\mathbf{j}) \in \mu \\ 0 & \text{otherwise} \end{cases} \quad ; \quad \mathbf{C}^\lambda_{\mathbf{j}} = \begin{cases} 1 & \mathbf{j} \in \lambda \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Then, given an input $\mathbf{A} \in \mathbb{R}^{n^k \times d}$, the output of an equivariant layer $L_{k \to l}$ can be written as follows:

$$L_{k \to l}(\mathbf{A})_{\mathbf{j}} = \sum_\mu \sum_{\mathbf{i}} \mathbf{B}^\mu_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w_\mu + \sum_\lambda \mathbf{C}^\lambda_{\mathbf{j}} b_\lambda, \tag{3}$$

where the learnable parameters are $w_\mu \in \mathbb{R}^{d \times d'}$ and $b_\lambda \in \mathbb{R}^{d'}$.

Let us consider the simplest case of linear layers defined above. Given a set of features $\mathbf{A} \in \mathbb{R}^{n \times d}$, the first-order equivariant linear layer can be written as:

$$L_{1 \to 1}(\mathbf{A}) = I_n \mathbf{A} W_1' + (1_n 1_n^\top - I_n)\mathbf{A} W_2 + 1_n b^\top \tag{4}$$

$$= I_n \mathbf{A} W_1 + 1_n 1_n^\top \mathbf{A} W_2 + 1_n b^\top, \tag{5}$$

where $W_1, W_2 \in \mathbb{R}^{d \times d'}$, $b \in \mathbb{R}^{d'}$. Notice that $L_{1 \to 1}$ is analogous to a combination of elementwise feedforward and sum-pooling of set elements. When tweaked a bit so that $1_n 1_n^\top$ is used instead of $1_n 1_n^\top - I_n$ (note: the basis spans the same function space), it is also known as a DeepSet layer [23].

Transformers generalize DeepSets As shown in the above section, the first-order (set-input) linear layers, or DeepSets, have rather simple structures composed of pooling and feedforward. Nevertheless, the architectures composed of the layers are proven to be universal approximators of permutation invariant functions [23]. Despite the theoretical advantage, static sum-pooling could be limited in capturing interactions between set elements, motivating the use of sophisticated pooling functions. In particular, it was shown that self-attention mechanism of Transformer encoder [18] achieves a large performance gain in set modeling tasks via context-aware weighted pooling [13, 22]. To see this, let us outline the operation of a Transformer encoder layer.

A Transformer encoder layer is a function $\text{Enc} : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$ consisting of two layers: a self-attention layer $\text{Attn} : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$ and an elementwise feedforward layer $\text{MLP} : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$. For a set of n input vectors $X \in \mathbb{R}^{n \times d}$, a Transformer layer computes the following:

$$\text{Attn}(X)_i = X_i + \sum_{h=1}^{H} \sum_{j=1}^{n} \alpha^h_{ij} X_j W^V_h W^O_h, \tag{6}$$

$$\text{Enc}(X)_i = \text{Attn}(X)_i + \text{MLP}(\text{Attn}(X))_i, \tag{7}$$

where $\alpha^h = \sigma(X W^Q_h (X W^K_h)^\top)$ is an attention coefficient with an activation σ, H is the number of heads, d_H is head size, d_F is hidden dimension, and $W^O_h \in \mathbb{R}^{d_H \times d}$, $W^V_h, W^K_h, W^Q_h \in \mathbb{R}^{d \times d_H}$.[2]

Now, we show that Transformer layers are generalized first-order linear equivariant functions. By setting $\alpha^h_{ij} = 1$ and assuming that $\text{MLP}(Y)$ approximates a linear layer $YW^F + b^{F\top}$ following universal approximation theorem [9], the Transformer layer reduces to the following[3]:

$$\text{Enc}(X)_i = X_i(I_n + W^F) + \sum_{j=1}^{n} X_j W^{VO}(I_n + W^F) + b^{F\top}, \tag{8}$$

where $W^{VO} = \sum_{h=1}^{H} W^V_h W^O_h$. This is equivalent to a DeepSet layer in Eq. (5) with $W^1 = I_n + W^F$, $W^2 = W^{VO}(I_n + W^F)$, and $b = b^F$. In other words, we can convert DeepSets to Transformers by changing its static pooling to attention and replacing elementwise linear mapping by an MLP. We generalize this approach to higer-order input and output, which leads to the formulation of higher-order Transformers for graphs and hypergraphs.

[2] Note that we omitted normalization after $\text{Attn}(\cdot)$ and $\text{MLP}(\cdot)$ for simplicity as in [22, 8].

[3] In practice, scaled softmax $\text{softmax}(\cdot/\sqrt{d_H})$ is often chosen as the non-linear activation $\sigma(\cdot)$; In this case, α^h_{ij} cannot be all one due to normalization and Transformers deviate from DeepSets.

3 Higher-order Transformer layers

In Section 2, we showed that Transformer layers are generalized first-order equivariant linear layers $L_{1\to1}$. Notably, the generalization procedure was equivalent to changing static pooling to attention and adding feedforward MLP. In this section, we generalize the approach to $L_{k\to l}$ with arbitrary orders k and l and propose higher-order Transformer layer $\text{Enc}_{k\to l}$.

In general, we define higher-order Transformer layer to be a function $\text{Enc}_{k\to l} : \mathbb{R}^{n^k\times d} \to \mathbb{R}^{n^l\times d}$ consisting of two layers: a higher-order self-attention layer $\text{Attn}_{k\to l} : \mathbb{R}^{n^k\times d} \to \mathbb{R}^{n^l\times d}$ and a feedforward layer $\text{MLP}_{l\to l} : \mathbb{R}^{n^l\times d} \to \mathbb{R}^{n^l\times d}$. For an input tensor $\mathbf{A} \in \mathbb{R}^{n^k\times d}$, a Transformer layer computes the following:

$$\text{MLP}_{l\to l}(\text{Attn}_{k\to l}(\mathbf{A})) = L^2_{l\to l}\left(\text{ReLU}(L^1_{l\to l}(\text{Attn}_{k\to l}(\mathbf{A})))\right), \tag{9}$$

$$\text{Enc}_{k\to l}(\mathbf{A}) = \text{Attn}_{k\to l}(\mathbf{A}) + \text{MLP}_{l\to l}(\text{Attn}_{k\to l}(\mathbf{A})), \tag{10}$$

where $L^1_{l\to l} : \mathbb{R}^{n^{l\times d}\to n^{l\times d_F}}$ and $L^2_{l\to l} : \mathbb{R}^{n^{l\times d_F}\to n^{l\times d}}$ are equivariant linear layers with hidden dimension d_F. Remaining question is how to define and compute higher-order self-attention $\text{Attn}_{k\to l}(\mathbf{A})$.

To design higher-order self-attention $\text{Attn}_{k\to l}$, we remove the bias term from Eq. (3) and introduce attention coefficients. This is done by changing each $\mathbf{B}^\mu \in \mathbb{R}^{n^{k+l}}$ to an attention score tensor $\boldsymbol{\alpha}^{h,\mu} \in \mathbb{R}^{n^{k+l}}$ with multiple heads as follows:

$$\text{Attn}_{k\to l}(\mathbf{A})_\mathbf{j} = \sum_{h=1}^{H}\sum_{\mu}\sum_{\mathbf{i}} \boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}}\mathbf{A}_\mathbf{i} w^V_{h,\mu} w^O_{h,\mu}, \tag{11}$$

where $w^O_{h,\mu} \in \mathbb{R}^{d_H\times d}$, $w^V_{h,\mu} \in \mathbb{R}^{d\times d_H}$ are learnable parameters, H denotes the number of heads, and d_H denotes head size. Then similar to Section 2, we can show the following:

Theorem 1. $\text{Enc}_{k\to l}$ *(Eq. (10)) is a strict generalization of* $L_{k\to l}$ *(Eq. (3)).*

Proof. Let $\boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}} = 1$ for all h, μ, and $(\mathbf{i},\mathbf{j}) \in \mu$. This renders $\boldsymbol{\alpha}^{h,\mu} = \mathbf{B}^\mu$ from the definition of $\mathbf{B}^\mu$. Additionally, let $\text{MLP}_{l\to l}(\text{Attn}_{k\to l}(\mathbf{A}))_\mathbf{j} = \sum_\lambda \mathbf{C}^\lambda_\mathbf{j} b_\lambda, \forall \mathbf{j} \in [n]^l$. That is, $\text{MLP}_{l\to l}$ ignores input and reduces to an invariant bias in Eq. (3). Then, Eq. (10) reduces to the following:

$$\text{Enc}_{k\to l}(\mathbf{A})_\mathbf{j} = \sum_{\mu}\sum_{\mathbf{i}} \mathbf{B}^\mu_{\mathbf{i},\mathbf{j}}\mathbf{A}_\mathbf{i} \sum_{h=1}^{H} w^V_{h,\mu} w^O_{h,\mu} + \sum_\lambda \mathbf{C}^\lambda_\mathbf{j} b_\lambda, \tag{12}$$

which is equivalent to Eq. (3) with $w_\mu = \sum_{h=1}^{H} w^V_{h,\mu} w^O_{h,\mu}$. □

Each attention tensor is obtained from higher-order query and key tensors as following:

$$\boldsymbol{\alpha}^\mu_{\mathbf{i},\mathbf{j}} = \begin{cases} \sigma(\mathbf{Q}^\mu_\mathbf{j}, \mathbf{K}^\mu_\mathbf{i})/Z_\mathbf{j} & (\mathbf{i},\mathbf{j}) \in \mu \\ 0 & \text{otherwise} \end{cases} \quad \text{where } \mathbf{Q}^\mu = L^\mu_{k\to l}(\mathbf{A}), \mathbf{K}^\mu = L^\mu_{k\to k}(\mathbf{A}). \tag{13}$$

where $Z_\mathbf{j} = \sum_\mathbf{i} \sigma(\mathbf{Q}^\mu_\mathbf{j}, \mathbf{K}^\mu_\mathbf{i})$ is a normalization constant. Note that the permutation invariance/equivariance are preserved in Transformer encoder in Eq. (10) since the attention coefficients are computed via permutation equivariant layers via Eq. (13).

4 Efficient Implementation of Higher-Order Transformers

In Section 3, we formulated the higher-order Transformer layers $\text{Enc}_{k\to l}$ and showed that they generalize linear equivariant layers $L_{k\to l}$. However, this capability comes with a cost; it turned out that the asymptotic complexity of the transformer encoder is $\mathcal{O}(n^{k+l})$ limiting its practical merits. In this section, we carefully analyze the encoder layer and show that much of the computations can be optimized, shared, and relaxed to reduce the complexity while minimizing its impact on capability. Notably, we show that combining all these strategies reduces the complexity to $\mathcal{O}(m)$ given a hypergraph with m hyperedges. Even with such efficiency, we show that the reduced version of higher-order Transformer is strictly more powerful than all message-passing neural networks.

Reducing orders of query and key First of all, although Eq. (11) and Eq. (13) provide a simple and generic definition of higher-order self-attention, we observe that there exists plenty of unnecessary computations. Specifically, many elements in query $\mathbf{Q}^\mu$ and key $\mathbf{K}^\mu$ are not used in computation of attention coefficient $\boldsymbol{\alpha}^\mu_{\mathbf{i},\mathbf{j}}$, as it depends only on indices satisfying $(\mathbf{i},\mathbf{j}) \in \mu$. In fact, it turns out that the effective orders of query and key are much smaller than l and k, as we show below:

Proposition 1. *From Eq. (13), let $u(\cdot)$ denote the number of unique entries in a multi-index. With $u_q = u(\mathbf{j})$, $u_k = u(\mathbf{i})$ for some $(\mathbf{i},\mathbf{j}) \in \mu$, we can always find suitable linear layers $L^\mu_{k\to u_q}$, $L^\mu_{k\to u_k}$ and index space mappings $f^\mu_q : [n]^l \to [n]^{u_q}$, $f^\mu_k : [n]^k \to [n]^{u_k}$ that satisfies the following.*

$$\boldsymbol{\alpha}^\mu_{\mathbf{i},\mathbf{j}} = \sigma(\tilde{\mathbf{Q}}^\mu_{\mathbf{j}'}, \tilde{\mathbf{K}}^\mu_{\mathbf{i}'})/\tilde{Z}_{\mathbf{j}'} \ \forall(\mathbf{i},\mathbf{j}) \in \mu, \tag{14}$$

where $\tilde{Z}_{\mathbf{j}} = \sum_{\mathbf{i}} \sigma(\tilde{\mathbf{Q}}^\mu_{\mathbf{j}'}, \tilde{\mathbf{K}}^\mu_{\mathbf{i}'})$, $\tilde{\mathbf{Q}}^\mu = L^\mu_{k\to u_q}(\mathbf{A})$, $\tilde{\mathbf{K}}^\mu = L^\mu_{k\to u_k}(\mathbf{A})$, $\mathbf{j}' = f^\mu_q(\mathbf{j})$, $\mathbf{i}' = f^\mu_k(\mathbf{i})$.

Proof. See the supplementary file. □

Proposition 3 implies that there exists much compact way to compute the query and key in Eq. (13) using linear layers of u_q and u_k orders. It not only reduces memory footprint and computations, but also provides a principled approach to share the parameters. Specifically,

Lightweight linear layers Although order reduction and query/key sharing improves efficiency, we observe that they do not help in reducing asymptotic computational complexity which is dominated by worst-cases $u_k = k$, $u_q = l$. To see this, we analyze the complexity of $\text{Enc}_{k\to l}$ (Eq. (10)) with respect to input size n. Starting from linear layers $L_{k\to l}$, we can show the following:

Property 1. *Given input size n, the asymptotic complexity of a linear layer $L_{k\to l}$ and the encoder layer $\text{Enc}_{k\to l}$ are $\mathcal{O}(n^{k+l})$ and $\mathcal{O}(n^{k+l} + n^{2k} + n^{2l})$, respectively.*

Proof. See the supplementary file. □

The major computation bottleneck is the higher-order linear layers, since it is used to compute the key, query, and MLP in the encoder. By exploiting only a subset of basis, we show that we can implement lightweight version of linear layer with reduced asymptotic complexity. Specifically, the optimization is based on following property:

Proposition 2. *Given a linear layer $L_{k\to l}$ in Eq. (3), we can always find a nonempty $M \subseteq [n]^{k+l}/_\sim$ such that computation of the following for all $\mathbf{j}$ has $\mathcal{O}(n^k + n^l)$ complexity.*

$$L_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu\in M} \sum_{\mathbf{i}} \mathbf{B}^\mu_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w_\mu + \sum_\lambda \mathbf{C}^\lambda_{\mathbf{j}} b_\lambda, \tag{15}$$

Proof. See the supplementary file. □

From Proposition 2, we define a lightweight linear layer $\bar{L}_{k\to l}$ as a linear layer in Eq. (15) with maximal M. By adopting $\bar{L}_{k\to l}$, the complexity of $L^\mu_{k\to u_k}$, $L^\mu_{l\to u_q}$, and $\text{MLP}_{l\to l}$ in Eq. (10) reduces to $\mathcal{O}(n^k)$, $\mathcal{O}(n^l)$, and $\mathcal{O}(n^l)$, respectively, and we have $\mathcal{O}(n^{k+l})$ in total for $\text{Enc}_{k\to l}$. Importantly, we show that this implementation does not hurt Theorem 1, because the attention coefficients can still reduce to one and MLP can still reduce to invariant bias. As a result, we have higher-order Transformer layers $\text{Enc}_{k\to l}$ that strictly generalize $L_{k\to l}$ with same complexity $\mathcal{O}(n^{k+l})$. From now on we assume that all linear layers for key, query, and MLP are lightweight.

Sparse Transformer layers Even with the approaches presented above, in practice $\mathcal{O}(n^{k+l})$ is already too large. For example, in graph domain, complexity larger than $\mathcal{O}(n^2)$ is regarded prohibitive while $\text{Enc}_{2\to 2}$ is still $\mathcal{O}(n^4)$. Thus, it is imperative that we further lower the complexity. To this end, we propose to exploit the sparsity inherent in real-world data. As we will show, this reduces the complexity of $\text{Enc}_{k\to l}$ to $\mathcal{O}(m^2)$ for processing a hypergraph with m edges (when $k \geq l$).

Often, along with order-k hypergraph $G = (V, \mathbf{A})$, the set of order-k hyperedges $E = \{\mathbf{i}_1, ..., \mathbf{i}_m\}$ where each $\mathbf{i} \in [n]^k$ is additionally given. E provides information on the tensor $\mathbf{A}$'s sparsity structure, that is, $\mathbf{A}_{\mathbf{i}}$ can be nonzero only if $\mathbf{i} \in E$. We will notate a sparse hypergraph data with hyperedges E as $G = (V, E, \mathbf{A})$, where often $m \ll n^k$.

A straightforward way of exploiting sparsity is to locally exchange information according to connectivity given as E, which is the approach adopted by message-passing neural networks of $k = 2$ and $\mathcal{O}(n + m)$ complexity [6, 11, 19]. Less intuitively, our question is how to incorporate E into computation of higher-order $\mathrm{Enc}_{k\to l}$ that contains a handful of global interactions. Here, we take a simple approach: constrain the input and output scope to the multi-indices defined by E.

Let us consider how we can constrain $L_{k\to l}$ given an order-k input $G = (V, E, \mathbf{A})$. As E specifies order-k connectivity, an immediate question is how to constrain the order-l output when $k \neq l$. To this end, we convert E accordingly to the output order. As lower-to-higher order conversion potentially results in globally dense structure (imagine $k = 1$ and $l = 2$, which is set-to-graph), we assume $k \geq l$. Then, a natural way to convert E to order-l is by replacing each $\mathbf{i} \in E$ with the maximal collection of $\mathbf{j} \in [n]^l$ such that every entry of each $\mathbf{j}$ is contained in $\mathbf{i}$. For example, with $k = 3$ and $l = 2$, this is equivalent to converting an input mesh into output graph by taking sides of each triangle. We denote the converted set of hyperedges as E'. Note that this conversion increases the size of E at most by a factor of $\binom{k}{l}$ and thus does not affect the asymptotic complexity.

Then, we define that $L_{k\to l}$ in Eq. (3) computes the following when given E:

$$L_{k\to l}(\mathbf{A}, E)_{\mathbf{j}} = \begin{cases} \sum_\mu \sum_{\mathbf{i}\in E} \mathbf{B}^\mu_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w_\mu + \sum_\lambda \mathbf{C}^\lambda_{\mathbf{j}} b_\lambda & \text{if } \mathbf{j} \in E' \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

We now define that $\mathrm{Enc}_{k\to l}$ in Eq. (10) computes the following when given E:

$$\mathrm{MLP}_{l\to l}(\mathrm{Attn}_{k\to l}(\mathbf{A}, E), E) = L^2_{l\to l}\left(\mathrm{ReLU}(L^1_{l\to l}(\mathrm{Attn}_{k\to l}(\mathbf{A}, E), E)), E\right), \tag{17}$$

$$\mathrm{Enc}_{k\to l}(\mathbf{A}, E) = \mathrm{Attn}_{k\to l}(\mathbf{A}, E) + \mathrm{MLP}_{l\to l}(\mathrm{Attn}_{k\to l}(\mathbf{A}, E), E), \tag{18}$$

where self-attention in Eq. (11) is computed as following when given E:

$$\mathrm{Attn}_{k\to l}(\mathbf{A}, E)_{\mathbf{j}} = \begin{cases} \sum_{h=1}^H \sum_\mu \sum_{\mathbf{i}\in E} \boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w^V_{h,\mu} w^O_{h,\mu} & \text{if } \mathbf{j} \in E' \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$\boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}} = \begin{cases} \sigma(\mathbf{Q}^{h,\mu}, \mathbf{K}^{h,\mu})_{\mathbf{i},\mathbf{j}} & \text{if } (\mathbf{i},\mathbf{j}) \in \mu, \mathbf{i} \in E, \mathbf{j} \in E' \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

where $\mathbf{Q}^{h,\mu} = L^{h,\mu}_{k\to l}(\mathbf{A}, E)$, $\mathbf{K}^{h,\mu} = L^{h,\mu}_{k\to k}(\mathbf{A}, E)$.

Now we show the following:

Property 2. *When given E with $m = |E|$, $L_{k\to l}$ has complexity $\mathcal{O}(m^2)$, $\bar{L}_{k\to l}$ has complexity $\mathcal{O}(m)$, and $\mathrm{Enc}_{k\to l}$ has complexity $\mathcal{O}(m^2)$.*

Proof. See the supplementary file. □

Kernel attention trick Analysis in the above paragraph shows a striking result that, only by constraining each components to have sparse input and output, we can reduce polynomial $\mathcal{O}(n^{k+l})$ complexity of $\mathrm{Enc}_{k\to l}$ to quadratic $\mathcal{O}(m^2)$ to input size. Yet, even $\mathcal{O}(m^2)$ can be demanding when input (hyper)graph gets large.

Here, we note that the quadratic complexity comes from computation and application of attention coefficients $\boldsymbol{\alpha}^\mu$. There is a similar problem in NLP domain, where $\mathcal{O}(n^2)$ complexity of first-order attention is prohibitive for long sequences. A line of works on efficient transformers tackle this problem by viewing attention coefficients as pairwise output of dot-product kernel [10, 3, 1]. As this allows sharing key-value computation for all queries, the complexity reduces to $\mathcal{O}(n)$. We adopt an analogous approach to change the computation order, which reduces the complexity of $\mathrm{Enc}_{k\to l}$ to $\mathcal{O}(n^k + n^l)$ and even $\mathcal{O}(m)$ for sparse version at the cost of relaxing some modeling assumption.

Let us assume that attention coefficient in Eq. (13) is computed as pairwise dot-product of feature mappings of query and key:

$$\boldsymbol{\alpha}^\mu_{\mathbf{i},\mathbf{j}} = \begin{cases} \phi(\mathbf{Q}^\mu_{\mathbf{j}})^\top \phi(\mathbf{K}^\mu_{\mathbf{i}}) & \text{if } (\mathbf{i},\mathbf{j}) \in \mu \\ 0 & \text{otherwise} \end{cases} \tag{21}$$

where $\phi : \mathbb{R}^{d_H} \to \mathbb{R}^{d_H}_+$ is kernel feature map. As attention scores are computed pairwise, we can apply Proposition 3 and write with lower-order query and key as in Eq. (27). With $\tilde{\mathbf{Q}}^\mu$, $\tilde{\mathbf{K}}^\mu$ compact query and key of order u_q, u_k, and lower-order indices $\mathbf{i}', \mathbf{j}' = f^\mu_k(\mathbf{i}), f^\mu_q(\mathbf{j})$, we can write:

$$\phi(\mathbf{Q}^\mu_{\mathbf{j}})^\top \phi(\mathbf{K}^\mu_{\mathbf{i}}) = \phi(\tilde{\mathbf{Q}}^\mu_{\mathbf{j}'})^\top \phi(\tilde{\mathbf{K}}^\mu_{\mathbf{i}'}) \; \forall (\mathbf{i},\mathbf{j}) \in \mu, \tag{22}$$

Now, let us recall the attention in Eq. (11). Dropping heads for brevity, we have:

$$\mathrm{Attn}_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu}\sum_{\mathbf{i}} \boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w^V_{\mu} w^O_{\mu}, \tag{23}$$

Here, we observe a critical problem: even with dot-product kernel assumption, it is impossible to share the attention computation over $\mathbf{i}$ across different $\mathbf{j}$. To understand this problem, let us recall $L_{1\to 1}$, specifically one of the equivalence classes $\mu = \{\{1\},\{2\}\}$. The equivalence class μ, by definition, restricts $i \neq j \forall (i,j) \in \mu$. This means, when computing j-th output entry, j-th input entry must be excluded from summation ($L_{1\to 1}(\mathbf{A})_j = \sum_{i\in[n]\setminus\{j\}} \mathbf{A}_i$). This *self-relation exclusion* property extends to higher-order linear layers, making it impossible to simply reuse the sum of input entries for different outputs. DeepSets and first-order Transformers are implicitly bypassing this problem by using the set of alternative basis $\hat{\mathbf{B}}^{\mu_1} = 1_n 1_n^\top$ and $\hat{\mathbf{B}}^{\mu_2} = I_n$ that span the same function space while decoupling the input and output indices. In this case, every input entries participate for computing j-th output, and the summation can be reused over outputs ($\hat{L}_{1\to 1}(\mathbf{A})_j = \sum_{i\in[n]} \mathbf{A}_i$). The implementation of $L_{2\to 2}$ by Maron et al. [2019] also shows that we can find similar alternative basis for second-order. However, it is unclear whether these findings that allows decoupling of $\mathbf{i}, \mathbf{j}$ also extend to higher-order.

To handle this problem, at the risk of losing some dimensionality, we choose to enforce decoupling of $\mathbf{i}, \mathbf{j}$ and share computation over different $\mathbf{j}$. This is done in two-step, which we write in below. First, using Eq. (21) and Eq. (22), we substitute $\boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}}$ by $\phi(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'})^\top \phi(\tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'})$ in Eq. (23):

$$\mathrm{Attn}_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu}\sum_{\mathbf{i}} \phi(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'})^\top \phi(\tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'}) \mathbf{A}_{\mathbf{i}} w_{\mu}, \tag{24}$$

where $w_\mu = w^V_\mu w^O_\mu$, $\mathbf{i}', \mathbf{j}' = f^{\mu}_k(\mathbf{i}), f^{\mu}_q(\mathbf{j})$. Then, we move the query out of the summation, obtaining the following:

$$\mathrm{Attn}_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu} \phi(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'})^\top \sum_{\mathbf{i}} \phi(\tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'}) \mathbf{A}_{\mathbf{i}} w_{\mu}, \tag{25}$$

Here, let us denote the output of inner summation as $M^{\mu}_{\mathbf{j}} = \sum_{\mathbf{i}} \phi(\tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'}) \mathbf{A}_{\mathbf{i}} w_{\mu}$. Strictly, $M^{\mu}_{\mathbf{j}}$ should be different for each $\mathbf{j}$, because we have coupling imposed by $(\mathbf{i},\mathbf{j}) \in \mu$. Here, we decide to share a same M^{μ} for all $\mathbf{j}$, thereby losing self-relation exclusion. In this case, we need to sum over the union of all summation scope of $\mathbf{i}$ required by $\mathbf{j}$, which we denote $\mathbf{i} \in \psi$:

$$\mathrm{Attn}_{k\to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu} \phi(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'})^\top \sum_{\mathbf{i}\in\psi} \phi(\tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'}) \mathbf{A}_{\mathbf{i}} w_{\mu} \tag{26}$$

Now we show the following:

Property 3. *The computation in Eq. (26) for all* $\mathbf{j}$ *has complexity* $\mathcal{O}(n^k + n^l)$. *When* E *is given, the computation has complexity* $\mathcal{O}(m)$ *with* $m = |E|$.

Proof. See the supplementary file. □

[todo: Introduce key normalization by multi-head attention against pseudo-queries]

Comparison to message passing As shown, by exploiting sparsity and adopting kernel attention we can reduce computational complexity of $\mathrm{Enc}_{k\to l}$ from $\mathcal{O}(n^{k+l})$ to $\mathcal{O}(m)$, linear to input size (at the cost of relaxing some modeling assumption). When we consider $k = l = 2$ (graph data), this complexity is equivalent or better than the $\mathcal{O}(n+m)$ complexity of message passing operation[4]. In this case, it is natural to particularly ask the expressiveness of $\mathrm{Enc}_{2\to 2}$ compared to message passing.

We first note that message passing cannot approximate some of the operations done by a single Transformer layer $\mathrm{Enc}_{2\to 2}$. Given a graph with diameter $d(E)$, we need at least $d(E)$ message passing operations to approximate output of $\mathrm{Enc}_{2\to 2}$. This is because a single $\mathrm{Enc}_{2\to 2}$ can impose dependency between any pair of input and output indices $\mathbf{i}, \mathbf{j}$ via M_μ, while message passing requires $d(E)$ steps in the worst case. Consequently, the approximation becomes impossible when $d(E) \to +\infty$, or when the graph contains > 1 disconnected components.

Now, we show the following:

[4] In practice, we place node features on the diagonals of the adjacency matrix, leading to $\mathcal{O}(n+m)$.

Theorem 2. *Given a graph $G = (V, E)$ with node features $\mathbf{X} \in \mathbb{R}^{n \times d_v}$ and edge features $\mathbf{E} \in \mathbb{R}^{n \times n \times d_e}$, a composition of two Transformer layers $\mathrm{Enc}_{2\to 2}$ with sparsity and kernel attention can approximate any message passing algorithm that fall into the framework of Gilmer et al. [2017] to an arbitrary precision.*

Proof. See the supplementary file. □

Theorem 2, combined with discussion on disconnected graphs, shows that a second-order Transformer composed of $\mathrm{Enc}_{2\to 2}$ can approximate any message-passing neural networks to arbitrary precision, and the opposite is not true. This leads to the following corollary:

Corollary 1. *Second-order Transformers are strictly more powerful than any message-passing neural networks within the framework of Gilmer et al. [2017].*

In particular, we note that local information propagation done in MPNNs is carried out in $\mathrm{Enc}_{2\to 2}$ by a single $\mu_1 = \{\{1\}, \{2, 3, 4\}\}$. Other types of μ would carry out different operations, which provides intuition on the powerfulness of Transformers.

5 Experiments

Implementation details We implemented higher-order Transformers in PyTorch [16]. The layers were implemented via efficient vectorization of the index-based formulation (see supplementary file). We implemented $\mathrm{Enc}_{k\to l}$ for $(k, l) = (1, 1), (1, 2), (1, 3), (2, 1), (2, 2)$, and sparse versions for $(k, l) = (1, 1), (2, 1), (2, 2)$. For sparse layers, we additionally implemented versions with kernel attention. [todo: explain normalization] For comparison, we implemented the linear layers $L_{k\to l}$ in Maron et al. [2019] and sparse versions as well. Our networks used in experiments are composition of $3 - 4$ layers, followed by invariant max-pooling [15] and MLP with $2 - 3$ hidden layers.

Synthetic datasets We tested higher-order Transformers on several synthetic tasks that highlight their differences against invariant MLPs and message-passing GNNs. [todo: setup]

Graph classification We tested our method on standard graph classification benchmark.

[todo: setup]

Set-to-graph prediction [todo: setup]

6 Discussion

[todo: Discuss about strength (theoretical completeness, inheritance of expressive powers of linear equivariant layers, ability to model global interactions, etc.) and limitation of our work (complexity of formulation and number of basis growing rapidly due to $O\left((\frac{0.792n}{\ln(n+1)})^n\right)$ asymptotic formula of n-th Bell number).]

References

Irwan Bello. Lambdanetworks: Modeling long-range interactions without attention. *arXiv*, 2021.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, 2020.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. *arXiv*, 2020.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *NIPS*, 2017.

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

Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. In *NeurIPS*, 2020.

Boris Hanin and Mark Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv*, 2017.

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*. OpenReview.net, 2017.

Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *ICML*, 2019.

Guohao Li, Matthias Müller, Ali K. Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *ICLR*, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *KDD*, 2018.

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

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *ICLR*, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *NeurIPS*, 2017.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *arXiv*, 2018.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv*, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? **[TODO]**
   (b) Did you describe the limitations of your work? **[TODO]**
   (c) Did you discuss any potential negative societal impacts of your work? **[TODO]**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? **[TODO]**
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? **[TODO]**
   (b) Did you include complete proofs of all theoretical results? **[TODO]**
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? **[TODO]**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? **[TODO]**
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? **[TODO]**
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? **[TODO]**
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? **[TODO]**
   (b) Did you mention the license of the assets? **[TODO]**
   (c) Did you include any new assets either in the supplemental material or as a URL? **[TODO]**
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? **[TODO]**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? **[TODO]**
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? **[TODO]**
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? **[TODO]**
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? **[TODO]**

A Appendix

A.1 Proof of things

Proposition 3. *From Eq. (13), let $u(\cdot)$ denote the number of unique entries in a multi-index. With $u_q = u(\mathbf{j})$, $u_k = u(\mathbf{i})$ for some $(\mathbf{i}, \mathbf{j}) \in \mu$, we can always find suitable linear layers $L^{\mu}_{k\to u_q}$, $L^{\mu}_{k\to u_k}$ and index space mappings $f^{\mu}_q : [n]^l \to [n]^{u_q}$, $f^{\mu}_k : [n]^k \to [n]^{u_k}$ that satisfies the following.*

$$\boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}} = \sigma(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'}, \tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'})/\tilde{Z}_{\mathbf{j}'} \ \forall(\mathbf{i}, \mathbf{j}) \in \mu, \tag{27}$$

where $\tilde{Z}_{\mathbf{j}} = \sum_{\mathbf{i}} \sigma(\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'}, \tilde{\mathbf{K}}^{\mu}_{\mathbf{i}'})$, $\tilde{\mathbf{Q}}^{\mu} = L^{\mu}_{k\to u_q}(\mathbf{A})$, $\tilde{\mathbf{K}}^{\mu} = L^{\mu}_{k\to u_k}(\mathbf{A})$, $\mathbf{j}' = f^{\mu}_q(\mathbf{j})$, $\mathbf{i}' = f^{\mu}_k(\mathbf{i})$.

Proof. We first note that the collection of all $\mathbf{i}$ such that $(\mathbf{i}, \mathbf{j}) \in \mu$ forms an equivalence class in $[n]^k/_{\sim}$, which we denote μ_k. Likewise, the collection of all $\mathbf{j}$ such that $(\mathbf{i}, \mathbf{j}) \in \mu$ forms an equivalence class $\mu_q \in [n]^l/_{\sim}$. Thus, to compute $\boldsymbol{\alpha}^{\mu}$ in Eq. (13), it is sufficient that we have $\mathbf{K}^{\mu}_{\mathbf{i}} \forall \mathbf{i} \in \mu_k$ and $\mathbf{Q}^{\mu}_{\mathbf{j}} \forall \mathbf{j} \in \mu_q$. For all $\mathbf{i} \in \mu_k$ we have $u_k = u(\mathbf{i}) = |\mu_k|$, because for all $a, b \in [k]$, $\mathbf{i}_a = \mathbf{i}_b$ iff a and b belong to a same set in μ_k. Analogously for all $\mathbf{j} \in \mu_q$ we have $u_q = u(\mathbf{j}) = |\mu_q|$.

Now, let us analyze and reduce the computation of $\mathbf{Q}^{\mu} = L^{\mu}_{k\to l}(\mathbf{A})$. $\mathbf{K}^{\mu}$ can be analyzed analogously by letting $l = k$. From Eq. (2) and Eq. (3), with $\mathbf{k} \in [n]^k$, $\alpha \in [n]^{k+l}/_{\sim}$, $\lambda \in [n]^l/_{\sim}$:

$$\mathbf{Q}^{\mu}_{\mathbf{j}} = \sum_{\alpha} \sum_{\mathbf{k}} \mathbf{B}^{\alpha}_{\mathbf{k},\mathbf{j}} \mathbf{A}_{\mathbf{k}} w_{\alpha} + \sum_{\lambda} \mathbf{C}^{\lambda}_{\mathbf{j}} b_{\lambda}, \tag{28}$$

$$\text{where } \mathbf{B}^{\alpha}_{\mathbf{k},\mathbf{j}} = 1 \text{ if } (\mathbf{k}, \mathbf{j}) \in \alpha, 0 \text{ else}; \ \mathbf{C}^{\lambda}_{\mathbf{j}} = 1 \text{ if } \mathbf{j} \in \lambda, 0 \text{ else} \tag{29}$$

A key observation here is that, when we want $\mathbf{Q}^{\mu}_{\mathbf{j}}$ only for $\mathbf{j} \in \mu_q$, a subset of basis in $L^{\mu}_{k\to l}$ only does redundant computation. Specifically, if a given α contains any $(\mathbf{k}, \mathbf{j})$ such that $\mathbf{j} \notin \mu_q$, then for all $\mathbf{k}$ and $\mathbf{j} \in \mu_q$, $\mathbf{B}^{\alpha}_{\mathbf{k},\mathbf{j}} = 0$. This is because all $(\mathbf{k}, \mathbf{j}) \in \alpha$ have the same equality pattern by definition; if one of such $\mathbf{j}$ is not contained in μ_q, then every others are not contained in μ_q as well. Therefore, we can safely discard such α.

Given that, how many α can we discarded from computation in Eq. (28) and how many should we keep? It turns out that the number of effective α is upper-bounded by $\mathrm{b}(k + u_q)$. To see this, recall that an effective α is one that satisfies $\mathbf{j} \in \mu_q \forall (\mathbf{k}, \mathbf{j}) \in \alpha$, because we want $\mathbf{Q}^{\mu}_{\mathbf{j}}$ only for $\mathbf{j} \in \mu_q$. As all $\mathbf{j} \in \mu_q$ contains exactly $u_q = |\mu_q|$ unique elements, within α, we always have $\{k + 1, ..., k + l\}$ contained in exactly u_q disjoint sets.

As every effective α must satisfy this condition, the number of effective α in Eq. (28) is upper-bounded by the number of ways of partitioning a set with $k + |\mu_q|$ elements, which is $\mathrm{b}(k + u_q)$. As for the bias, we can repeat the analysis with $k = 0$ which gives us the number of effective λ upper-bounded by $\mathrm{b}(u_q)$.

Now, let us denote $\mathcal{A}$ the collection of effective α with $\leq \mathrm{b}(k + u_q)$ elements and $\mathcal{L}$ the collection of effective λ with $\leq \mathrm{b}(u_q)$ elements. Then we can rewrite Eq. (28) as:

$$\mathbf{Q}^{\mu}_{\mathbf{j}} = \sum_{\alpha\in\mathcal{A}} \sum_{\mathbf{k}} \mathbf{B}^{\alpha}_{\mathbf{k},\mathbf{j}} \mathbf{A}_{\mathbf{k}} w_{\alpha} + \sum_{\lambda\in\mathcal{L}} \mathbf{C}^{\lambda}_{\mathbf{j}} b_{\lambda}, \tag{30}$$

Assume we have some $L^{\mu}_{k\to u_q}$. With $\mathbf{j}' \in [n]^{u_q}$, $\beta \in [n]^{k+u_q}/_{\sim}$, $\theta \in [n]^{u_q}/_{\sim}$, we can write:

$$\tilde{\mathbf{Q}}^{\mu}_{\mathbf{j}'} = \sum_{\beta} \sum_{\mathbf{k}} \mathbf{B}^{\beta}_{\mathbf{k},\mathbf{j}'} \mathbf{A}_{\mathbf{k}} w_{\beta} + \sum_{\theta} \mathbf{C}^{\theta}_{\mathbf{j}'} b_{\theta}, \tag{31}$$

$$\text{where } \mathbf{B}^{\beta}_{\mathbf{k},\mathbf{j}'} = 1 \text{ if } (\mathbf{k}, \mathbf{j}') \in \beta, 0 \text{ else}; \ \mathbf{C}^{\theta}_{\mathbf{j}'} = 1 \text{ if } \mathbf{j}' \in \theta, 0 \text{ else} \tag{32}$$

We now identify the condition that $\tilde{\mathbf{Q}}^{\mu}$ contains all $\mathbf{Q}^{\mu}_{\mathbf{j}} \forall \mathbf{j} \in \mu_q$ in Eq. (28).

To this end, we define a surjective mapping $g : [k] \to [u_q]$ that satisfies $\mathbf{j}_a = \mathbf{j}_b \Leftrightarrow g(a) = g(b)$. This is always possible, as for all $a, b \in [l]$, $\mathbf{j}_a = \mathbf{j}_b$ iff a, b belong to a same set in μ_q. By indexing each set in μ_q, for $a \in [l]$ we can define $g(a)$ as the index of the set in μ_q that a belongs.

Using g, for each $\mathbf{j} \in \mu_q$, we can find its compact form $\mathbf{j}' \in [n]^{u_q}$ that contains u_q unique elements. More specifically, $\mathbf{j}'$ is given by $\mathbf{j}'_c = \mathbf{j}_a = \mathbf{j}_b \forall a, b \in [l]$ with $c = g(a) = g(b)$. We denote $f^q_{\mu} : [n]^q \to [n]^{u_q}$ a mapping that gives $\mathbf{j}' = f^q_{\mu}(\mathbf{j}) \forall \mathbf{j} \in \mu_q$.

Now, we have all requirements to modify Eq. (28) into Eq. (31). First, for each $\alpha \in \mathcal{A}$, we can find a distinct $\beta \in [n]^{k+u_q}/_\sim$ such that, with $\mathbf{j}' = f^q_\mu(\mathbf{j})$, $(\mathbf{k}, \mathbf{j}') \in \beta \forall (\mathbf{k}, \mathbf{j}) \in \alpha$. We can similarly find distinct θ for each $\lambda \in \mathcal{L}$. Now, for all paired α and β, we set $w_\alpha = w_\beta$. We set $w_\beta = 0$ for every β not paired with any α. We similarly set $b_\theta = b_\lambda$ for each θ paired with λ, and $b_\theta = 0$ for θ not paired with any λ. From the definition of basis tensors, we have $\mathbf{B}^\alpha_{\mathbf{k},\mathbf{j}} = \mathbf{B}^\beta_{\mathbf{k},\mathbf{j}'}$ for all paired α and β, and $\mathbf{C}^\lambda_{\mathbf{j}} = \mathbf{C}^\theta_{\mathbf{j}'}$ for all paired λ and θ. Therefore, we have $\tilde{\mathbf{Q}}^\mu_{\mathbf{j}'} = \mathbf{Q}^\mu_{\mathbf{j}}$ for all $\mathbf{j} \in \mu_q$.

Thus, we can always find $L^\mu_{k \to u_q}$ and $f^\mu_q : [n]^l \to [n]^{u_q}$ sufficient to compute $\mathbf{Q}^\mu_{\mathbf{j}} \forall \mathbf{j} \in \mu_q$. As noted in the beginning of the proof, we can perform the same analysis with $k = l$ to show the analogous result for $\mathbf{K}^\mu$. We now have the sufficient $\mathbf{K}^\mu_{\mathbf{i}} \forall \mathbf{i} \in \mu_k$ and $\mathbf{Q}^\mu_{\mathbf{j}} \forall \mathbf{j} \in \mu_q$ to compute Eq. (13) and therefore Proposition 3 holds.

Caution: Note that $\tilde{\mathbf{Q}}$ still contains some redundant entries. As each entry of $\mathbf{j}'$ is unique by definition, all $\mathbf{j}'$ belong to a single equivalence class $\{\{1\}, \{2\}, ..., \{u_q\}\} \in [n]^{u_q}/_\sim$. But by using $L_{k \to u_q}$ to compute $\tilde{\mathbf{Q}}$, we are using other equivalence classes as well. Nevertheless, the number of such indices is $\mathcal{O}(n^{u_q - 1})$, which is smaller than $\tilde{\mathbf{Q}}$ by a factor of $1/n$ and becomes negligible with large n. □

Property 4. *Given input size n, The asymptotic complexity of a linear layer $L_{k \to l}$ is $\mathcal{O}(n^{k+l})$. The complexity of* $\mathrm{Enc}_{k \to l}$ *is $\mathcal{O}(n^{2k} + n^{k+l} + n^{2l})$, with n^{2k} term coming from key computation and n^{2l} term coming from query computation and* $\mathrm{MLP}_{l \to l}$ *after attention.*

Proof. We begin by analyzing the asymptotic complexity of an equivariant linear layer $L_{k \to l}$ (Eq. (3)). In the inner summation $\sum_{\mathbf{i}} \mathbf{B}^\mu_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}}$, with u_k and u_q the number of unique entries in $\mathbf{i}$ and $\mathbf{j}$ respectively, we have n^{u_k} effective multiplication and summations for each output index $\mathbf{j}$, and thus the number of operations done by the summation is $n^{u_q} n^{u_k}$. With outer summation over μ, we have $\sum_\mu n^{u_q} n^{u_k}$ operations. As $u_q \leq l$ and $u_k \leq k$ by definition, we have inequality $\sum_\mu n^{u_q} n^{u_k} \leq \mathrm{b}(k+l) n^{k+l}$. Application of w_μ gives us $dd' \sum_\mu n^{u_q} n^{u_k} \leq \mathrm{b}(k+l) dd' n^{k+l}$ number of operations. For the bias, in the inner term $\mathbf{C}^\lambda_{\mathbf{j}}$, we need a single addition for each $\mathbf{j}$ and thus the number of operations for a λ is $n^{u_q} \leq n^l$. Summation over λ and application of bias parameters gives us $\mathrm{b}(l) d' n^{u_q} \leq \mathrm{b}(l) d' n^l$ operations. Collectively, we have $\leq \mathrm{b}(k+l) dd' n^{k+l} + \mathrm{b}(l) d' n^l$ number of operations. As k, l, d, d' are constants that does not depend on n, we obtain an asymptotic complexity $\mathcal{O}(n^{k+l})$.

Computation of $\mathrm{Enc}_{k \to l}(\mathbf{A})$ (Eq. (10)) involves computing $\mathrm{Attn}_{k \to l}(\mathbf{A})$, $\mathrm{MLP}_{l \to l}(\mathrm{Attn}_{k \to l}(\mathbf{A}))$ and adding them. Let us analyze $\mathrm{Attn}_{k \to l}(\mathbf{A})$ first. To compute $\boldsymbol{\alpha}^{h,\mu}$ from input, we need to compute $L^\mu_{k \to u_q}(\mathbf{A})$ and $L^\mu_{k \to u_k}(\mathbf{A})$, followed by pairwise similarity computation and re-indexing. Assuming that pairwise similarity computation and indexing can be done with constant number of computations[5], we have asymptotic complexity $\mathcal{O}(n^{k+u_q} + n^{k+u_k} + n^{u_q+u_k})$. As $u_q \leq l$ and $u_k \leq k$, we have $\mathcal{O}(n^{k+l} + n^{2k})$. Here, note that the computation of keys from input gives rise to $\mathcal{O}(n^{2k})$. Having computed $\boldsymbol{\alpha}^{h,\mu}$, the inner summation $\sum_{\mathbf{i}} \boldsymbol{\alpha}^{h,\mu}_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}}$, similar to in $L_{k \to l}$, has $n^{u_q} n^{u_k}$ computations. With outer summation over μ, we have $\sum_\mu n^{u_q} n^{u_k} \leq \mathrm{b}(k+l) n^{k+l}$ operations. Summation over heads and application of weight matrices gives us $H d_H^2 d \sum_\mu n^{u_q} n^{u_k} \leq \mathrm{b}(k+l) H d_H^2 d n^{k+l}$ number of operations, which is $\mathcal{O}(n^{k+l})$. For application of $\mathrm{MLP}_{l \to l}(\cdot)$, we sum the complexity of two linear layers $L_{l \to l}$ and element-wise ReLU, which gives us asymptotic complexity of $\mathcal{O}(n^{2l})$. In summary, the asymptotic complexity of $\mathrm{Enc}_{k \to l}$ is $\mathcal{O}(n^{2k} + n^{k+l} + n^{2l})$. □

Proposition 4. *Given a linear layer $L_{k \to l}$ in Eq. (3), we can always find a nonempty $M \subseteq [n]^{k+l}/_\sim$ such that computation of the following has $\mathcal{O}(n^k + n^l)$ complexity.*

$$L_{k \to l}(\mathbf{A})_{\mathbf{j}} = \sum_{\mu \in M} \sum_{\mathbf{i}} \mathbf{B}^\mu_{\mathbf{i},\mathbf{j}} \mathbf{A}_{\mathbf{i}} w_\mu + \sum_\lambda \mathbf{C}^\lambda_{\mathbf{j}} b_\lambda, \tag{33}$$

Proof. among the basis tensors $\mathbf{B}^\mu$ of a linear layer $L_{k \to l}$, we select a subset that does not contain pooling over input elements, which reduces complexity to $\mathcal{O}(n^k)$. Applying this modification, the complexity of key computation and feedforward MLP reduces to $\mathcal{O}(n^k)$ and $\mathcal{O}(n^l)$ respectively and

[5] This does not hold when we normalize over keys, which gives an additive complexity $\mathcal{O}(n^{u_k})$.

we have $\mathcal{O}(n^{k+l})$ in total. As a side-effect, we have smaller set of basis that benefits efficiency[6]. Note that this design choice does not hurt the results of Proposition **??**, because the attention coefficients can still reduce to one and MLP can still reduce to invariant bias. □

Property 5. *The computation of* $\text{Enc}_{k\to l}$*, when given* E*, has complexity* $\mathcal{O}(m^2)$ *where* $m = |E|$.

Proof. Now, we analyze the asymptotic computational complexity of $\text{Enc}_{k\to l}$ when given E. We begin from equivariant linear layer $L_{k\to l}$ (Eq. (16)) when given E. In the inner summation $\sum_{\mathbf{i}\in E} \mathbf{B}^{\mu}_{\mathbf{i},\mathbf{j}}\mathbf{A}_{\mathbf{i}}$, the number of multiplication and addition for each $\mathbf{j}$ is upper-bounded by $m = |E|$. As the output scope is bounded by $m\binom{k}{l}$, the effective number of operations are $\leq m^2\binom{k}{l}$. With outer summation over μ, we have $\leq \text{b}(k+l)\binom{k}{l}m^2$ operations. As a result, we obtain a complexity $\mathcal{O}(m^2)$. Often in $\text{Enc}_{k\to l}$, we utilize lightweight linear layers that precludes summation over input (presented in Section 3). In this case we trivially have $\mathcal{O}(m)$ complexity as we do not sum over $\mathbf{i}$.

Computation of $\text{Enc}_{k\to l}(\mathbf{A}, E)$ (Eq. (18)) involves adding the outputs of $\text{Attn}_{k\to l}(\mathbf{A}, E)$ and $\text{MLP}_{l\to l}(\text{Attn}_{k\to l}(\mathbf{A}, E), E)$. Let us analyze $\text{Attn}_{k\to l}(\mathbf{A}, E)$ first. To compute $\boldsymbol{\alpha}^{\mu}$ from input, we need to compute lightweight linear layers $L^{\mu}_{k\to u_q}(\mathbf{A}, E)$ and $L^{\mu}_{k\to u_k}(\mathbf{A}, E)$, followed by pairwise similarity computation of nonzero entries. As $u_q, u_k \leq k$, we have complexity $\mathcal{O}(m)$ for the linear layers and $\mathcal{O}(m^2)$ for pairwise computation. Having computed $\boldsymbol{\alpha}^{\mu}$, the inner summation $\sum_{\mathbf{i}\in E} \boldsymbol{\alpha}^{\mu}_{\mathbf{i},\mathbf{j}}\mathbf{A}_{\mathbf{i}}$ has $\leq m$ computations. Enumerating over $\mathbf{j}$, we have $\mathcal{O}(m^2)$. For application of $\text{MLP}_{l\to l}$, we sum the complexity of two lightweight linear layers $L_{l\to l}$ and element-wise ReLU given E, which gives us $\mathcal{O}(m)$. In summary, the complexity of $\text{Enc}_{k\to l}$ when given E is $\mathcal{O}(m^2)$. □

Property 6. *The computation in Eq. (26) for all* $\mathbf{j}$ *has complexity* $\mathcal{O}(n^k + n^l)$*. When* E *is given, the computation has complexity* $\mathcal{O}(m)$ *with* $m = |E|$.

Proof. Summation over $\mathbf{i} \in \psi$ decouples the scope of $\mathbf{i}$ from $\mathbf{j}$ and allows reuse of computation over $\mathbf{j}$. As the summation over $\mathbf{i} \in \psi$ involves $\mathcal{O}(n^k)$ operations and we share it over all query indices $\mathbf{j}$, self-attention reduces to elementwise application and we obtain $\mathcal{O}(n^k + n^l)$ complexity. As computation of $\tilde{\mathbf{Q}}^{\mu}$, $\tilde{\mathbf{K}}^{\mu}$ and application of $\text{MLP}_{l\to l}$ are $\mathcal{O}(n^k + n^l)$ with lightweight linear layers, we have $\mathcal{O}(n^k + n^l)$ collective complexity for $\text{Enc}_{k\to l}$. When applied in conjunction with a sparse version that also receives E, summation over $\mathbf{i}$ and enumeration over $\mathbf{j}$ all reduce to $\mathcal{O}(m)$ and we thereby have $\mathcal{O}(m)$ complexity. □

Proposition 5. *Given a graph* $G = (V, E)$ *with node features* $\mathbf{X} \in \mathbb{R}^{n\times d_v}$ *and edge features* $\mathbf{E} \in \mathbb{R}^{n\times n\times d_e}$ *, a composition of two Transformer layers* $\text{Enc}_{2\to 2}$ *with sparsity and kernel attention can approximate any message passing algorithm that fall into the framework of Gilmer et al. [2017] to an arbitrary precision.*

Proof. With message function $M : \mathbb{R}^{2d_v+d_e} \to \mathbb{R}^{d_m}$ and update function $U : \mathbb{R}^{d_v+d_m} \to \mathbb{R}^{d}$, a message passing step outputs node features $\mathbf{H} \in \mathbb{R}^{n\times d}$ according to following. With $i, j \in [n]$:

$$\mathbf{M}_j = \sum_{i\in\mathcal{N}(j)} M(\mathbf{X}_j, \mathbf{X}_i, \mathbf{E}_{ij}) \tag{34}$$

$$\mathbf{H}_j = U(\mathbf{X}_j, \mathbf{M}_j), \tag{35}$$

where $\mathcal{N}(j)$ denotes incoming neighbors of j-th node or $\{i|(i, j) \in E\}$.

We now show how a composition of two $\text{Enc}_{2\to 2}$ can approximate above computation.

1. As a first step, we encode $\mathbf{X}$ and $\mathbf{E}$ into a single $\mathbf{A} \in \mathbb{R}^{n\times n\times(2d_v+d_e)}$ [15]. In the first d_v channels, we replicate $\mathbf{X}$ on the rows. In the next d_v channels, we replicate $\mathbf{X}$ on the columns. In the last d_e channels, we put $\mathbf{E}$. Additionally, to account for output positions (node features), we augment E with self-loops and make $E' = E \cup \{(ii)\forall i \in [n]\}$.

2. Then, we make the first $\text{Enc}_{2\to 2}$ approximate the message function $M(\cdot)$, so that $\text{Enc}_{2\to 2}(\mathbf{A})_{ij} \approx M(\mathbf{X}_j, \mathbf{X}_i, \mathbf{E}_{ij})$. To do this, we first reduce $\text{Attn}_{2\to 2}(\mathbf{A})_{ij} = \mathbf{A}_{ij}$ and apply $\text{MLP}_{l\to l}$ on top of it. We reduce $\text{MLP}_{l\to l}$ to entry-wise MLP. As $\text{Attn}_{2\to 2}(\mathbf{A})_{ij} = \mathbf{A}_{ij}$

[6]For $L_{k\to l}$, size of basis subset without pooling is equal to number of $\mu \in [n]^{k+l}/_{\sim}$ such that every set in μ containing ≥ 1 element in $\{1, ..., k\}$ also contains ≥ 1 element in $\{k + 1, ..., k + l\}$.

is a concatenation of $\mathbf{X}_i$, $\mathbf{X}_j$, $\mathbf{E}_{ij}$, with universal approximation theorem [9], we have the output of the first $\text{Enc}_{2\to2}(\mathbf{A})_{ij} = M(\mathbf{X}_j, \mathbf{X}_i, \mathbf{E}_{ij}) + \epsilon_1 \forall i, j$. This also holds when we restrict the index scope to $(i, j) \in E'$; we make $(i, j) \in E' \setminus E$ contain zero vectors.

3. Before feeding the output to the second $\text{Enc}_{2\to2}$, we concatenate the original input $\mathbf{A}$ with the output of the first layer to make $\mathbf{A}'$. This gives $\mathbf{X}_i, \mathbf{X}_j, \mathbf{E}_{ij}$ encoded in the first $2d_v + d_e$ channels and $M(\mathbf{X}_j, \mathbf{X}_i, \mathbf{E}_{ij}) + \epsilon_1$ encoded in the last d_m channels of $\mathbf{A}'$. This operation can be trivially absorbed within the MLP of the first layer, but we separate for simplicity.

4. Now, we make the second $\text{Enc}_{2\to2}$ jointly approximate summation of messages over neighbors $\sum_{i\in\mathcal{N}(j)}(\cdot)$ and update function $U(\cdot)$, so that $\text{Enc}_{2\to2}(\mathbf{A}')_{jj} \approx \mathbf{H}_j = U(\mathbf{X}_j, \mathbf{M}_j)$. First, we reduce $\text{Attn}_{2\to2}(\mathbf{A}')$ to summation over neighbors. For this we only need two equivalence classes $\mu_1 = \{\{1\}, \{2, 3, 4\}\}$ and $\mu_2 = \{\{1, 2, 3, 4\}\}$. We can write Eq. (26) as follows, with $u_q = 1$, $u_k = 2$, $\mathbf{i} = (i, j)$, $\mathbf{j} = (j, j)$, $\mathbf{i}' = (i, j)$, $\mathbf{j}' = j$ for μ_1 and $u_q = 1$, $u_k = 1$, $\mathbf{i} = (j, j)$, $\mathbf{j} = (j, j)$, $\mathbf{i}' = j$, $\mathbf{j}' = j$ for μ_2:

$$\text{Attn}_{2\to2}(\mathbf{A}')_{jj} = \phi(\tilde{\mathbf{Q}}_j^{\mu_1})^\top \sum_{\{i|(i,j)\in E'\}} \phi(\tilde{\mathbf{K}}_{ij}^{\mu_1})\mathbf{A}'_{ij} w_{\mu_1} + \phi(\tilde{\mathbf{Q}}_j^{\mu_2})^\top \phi(\tilde{\mathbf{K}}_j^{\mu_2})\mathbf{A}'_{jj} w_{\mu_2}. \tag{36}$$

Let entries in $\phi(\tilde{\mathbf{K}}^\mu)$, $\phi(\tilde{\mathbf{Q}}^\mu)$ be $\frac{1_{d_H}}{\sqrt{d_H}}$ so that their dot product is 1. Eq. (36) reduces to:

$$\text{Attn}_{2\to2}(\mathbf{A}')_{jj} = \sum_{\{i|(i,j)\in E'\}} \mathbf{A}'_{ij} w_{\mu_1} + \mathbf{A}'_{jj} w_{\mu_2} \tag{37}$$

We make w_{μ_1} zero-out the first $2d_v + d_e$ channels and w_{μ_2} zero-out the last $d_e + d_m$ channels. Then, we have $\text{Attn}_{2\to2}(\mathbf{A}')_{jj}$ contain $\mathbf{X}_j$ in the first d_v channels and $\mathbf{M}_j + \epsilon_2 = \sum_{i|(i,j)\in E'}(M(\mathbf{X}_j, \mathbf{X}_i, \mathbf{E}_{ij})) + \epsilon_2$ in the last d_m channels[7]. We then apply $\text{MLP}_{l\to l}$ on top of it, which can approximate the update function by universal approximation theorem [9] and we have $\text{Attn}_{2\to2}(\mathbf{A}')_{jj} = U(\mathbf{X}_j, \mathbf{M}_j) + \epsilon_3$.

Overall, the approximation error ϵ_i at each step depends on ϵ_{i-1} $(i > 1)$, the MLP that approximates relevant function, and uniform bounds and uniform continuity of the approximated functions. □

A.2 Vectorized Transformer encoder layers

Here, we exhaustively characterize $\text{Enc}_{k\to l}$ for $(k, l) \in \{(1, 1), (1, 2), (2, 1), (2, 2)\}$. For that, let us define some notations. We denote $\text{diagonal}(\cdot)$ as a matrix operation that selects diagonal entries, and $\text{diagonal}_{\text{axis}_1,\text{axis}_2}(\cdot)$ as a tensor operation that selects diagonal entries across axis_1, axis_2 and arranges them across axis_1. In the opposite, we denote $\text{diag}(\cdot)$ a vector operation that outputs a diagonal square matrix.

First-Order $(1 \to 1)$ The first-order equivariant $L_{1\to1} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d'}$ can be written as

$$L_{1\to1}(\mathbf{A}) = \mathbf{A}W^1 + \mathbf{M}^{(2)}\mathbf{A}W^2, \tag{38}$$

where $W^1, W^2 \in \mathbb{R}^{d\times d'}$ are learnable parameters. Extending $L_{1\to1}$, we define:

$$\text{SelfAtt}_{1\to1}(\mathbf{A}) = \mathbf{A}W^1 + \alpha^{(1,1)}\mathbf{A}W^2. \tag{39}$$

Mixed-Order $(1 \to 2)$ The mixed-order equivariant $L_{1\to2} : \mathbb{R}^{n\times d} \to \mathbb{R}^{n^2\times d'}$ can be written as:

$$L_{1\to2}(\mathbf{A}) = \text{diag}(\mathbf{A})W^1 + (\mathbf{A}\mathbf{1}^\top)^\top W^2 + \mathbf{A}\mathbf{1}^\top W^3 + \text{diag}(\mathbf{M}^{(2)}\mathbf{A})W^4 + \mathbf{M}^{(2,1)}\mathbf{A}W^5, \tag{40}$$

where $W^1, ..., W^5 \in \mathbb{R}^{d\times d'}$ are learnable parameters. Extending $L_{1\to2}$, we define:

$$\text{SelfAtt}_{1\to2}(\mathbf{A}) = \text{diag}(\mathbf{A})W^1 + (\mathbf{A}\mathbf{1}^\top)^\top W^2 + \mathbf{A}\mathbf{1}^\top W^3 + \text{diag}(\alpha^{(1,1)}\mathbf{A})W^4 + \alpha^{(2,1)}\mathbf{A}W^5. \tag{41}$$

[7] Note that message summation over $\{i|(i, j) \in E'\}$ is equivalent to summation over $\{i|(i, j) \in E\} = \mathcal{N}(j)$ because we set message zero at $(i, j) \in E' \setminus E$.

Mixed-Order $(2 \to 1)$ The mixed-order equivariant $L_{2\to1} : \mathbb{R}^{n^2\times d} \to \mathbb{R}^{n\times d'}$ can be written as:

$$\begin{aligned} L_{2\to1}(\mathbf{A}) = &\text{diagonal}(\mathbf{A})W^1 + \text{diagonal}(\mathbf{M}^{(2)}\mathbf{A}^\top)W^2 + \text{diagonal}(\mathbf{M}^{(2)}\mathbf{A})W^3 \\ &+ \mathbf{M}^{(2)}\text{diagonal}(\mathbf{A})W^4 + \mathbf{M}^{(1,2)}\mathbf{A}W^5, \end{aligned} \tag{42}$$

where $W^1, ..., W^5 \in \mathbb{R}^{d\times d'}$ are learnable parameters. Extending $L_{2\to1}$, we define:

$$\begin{aligned} \text{SelfAtt}_{2\to1}(\mathbf{A}) = &\text{diagonal}(\mathbf{A})W^1 + \text{diagonal}(\alpha^{(1,1)}\mathbf{A}^\top)W^2 + \text{diagonal}(\alpha^{(1,1)}\mathbf{A})W^3 \\ &+ \alpha^{(1,1)}\text{diagonal}(\mathbf{A})W^4 + \alpha^{(1,2)}\mathbf{A}W^5. \end{aligned} \tag{43}$$

Second-Order $(2 \to 2)$ The second-order equivariant $L_{2\to2} : \mathbb{R}^{n^2\times d} \to \mathbb{R}^{n^2\times d'}$ can be written as:

$$\begin{aligned} L_{2\to2}(\mathbf{A}) = &\mathbf{I} \odot \mathbf{A}W^1 + (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot \mathbf{A}W^2 + (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot \mathbf{A}^\top W^3 \\ &+ (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot (\text{diagonal}(\mathbf{A})\mathbf{1}^\top)^\top W^4 + (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot (\text{diagonal}(\mathbf{A})\mathbf{1}^\top)W^5 \\ &+ \mathbf{I} \odot (\mathbf{M}^{(2)}\mathbf{A})W^6 + \mathbf{I} \odot (\mathbf{M}^{(2)}\mathbf{A}^\top)W^7 \\ &+ \text{diagonal}_{2,3}(\mathbf{M}^{(2,1)}\mathbf{A})W^8 + \text{diagonal}_{1,3}(\mathbf{M}^{(2,1)}\mathbf{A}^\top)W^9 \\ &+ \text{diagonal}_{2,3}(\mathbf{M}^{(2,1)}\mathbf{A}^\top)W^{10} + \text{diagonal}_{1,3}(\mathbf{M}^{(2,1)}\mathbf{A})W^{11} \\ &+ \text{diag}(\mathbf{M}^{(2)}\text{diagonal}(\mathbf{A}))W^{12} + \text{diag}(\mathbf{M}^{(1,2)}\mathbf{A})W^{13} \\ &+ \mathbf{M}^{(2,1)}\text{diagonal}(\mathbf{A})W_{14} + \mathbf{M}^{(2,2)}\mathbf{A}W_{15}, \end{aligned} \tag{44}$$

where $W^1, ..., W^{15} \in \mathbb{R}^{d\times d'}$ are learnable parameters. Extending $L_{2\to2}$, we define:

$$\begin{aligned} \text{SelfAtt}_{2\to2}(\mathbf{A}) = &\mathbf{I} \odot \mathbf{A}W^1 + (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot \mathbf{A}W^2 + (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot \mathbf{A}^\top W^3 \\ &+ (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot (\text{diagonal}(\mathbf{A})\mathbf{1}^\top)^\top W^4 + (\mathbf{I} - \mathbf{1}\mathbf{1}^\top) \odot (\text{diagonal}(\mathbf{A})\mathbf{1}^\top)W^5 \\ &+ \mathbf{I} \odot (\alpha^{(1,1)}\mathbf{A})W^6 + \mathbf{I} \odot (\alpha^{(1,1)}\mathbf{A}^\top)W^7 \\ &+ \text{diagonal}_{2,3}(\alpha^{(2,1)}\mathbf{A})W^8 + \text{diagonal}_{1,3}(\alpha^{(2,1)}\mathbf{A}^\top)W^9 \\ &+ \text{diagonal}_{2,3}(\alpha^{(2,1)}\mathbf{A}^\top)W^{10} + \text{diagonal}_{1,3}(\alpha^{(2,1)}\mathbf{A})W^{11} \\ &+ \text{diag}(\alpha^{(1,1)}\text{diagonal}(\mathbf{A}))W^{12} + \text{diag}(\alpha^{(1,2)}\mathbf{A})W^{13} \\ &+ \alpha^{(2,1)}\text{diagonal}(\mathbf{A})W_{14} + \alpha^{(2,2)}\mathbf{A}W_{15}. \end{aligned} \tag{45}$$

A.3 Vectorized Sparse Transformer layers

In the main text, we developed dense tensor transformers that operates on dense (hyper-)graphs. Although dense transformers could potentially model global node interactions better than message-passing neural networks, there are critical drawbacks. Most of all, with dense input and output, the network has a prohibitive memory requirement for storing attention coefficient tensors (an encoder layer with order-2 input and output requires $\mathcal{O}(n^4)$ memory). One could also argue that the network is unable to leverage sparse topology of input graphs efficiently Dwivedi and Bresson [2021]. In this section, we deal with the aforementioned problems by developing a variant of the tensor transformer that leverages the sparse structure of input graphs to reduce memory requirement. In order to do so, we start from a simple assumption that, for equivariant layers, the topology of input and output (hyper-)graphs remains the same. From that, we derive the sparse tensor transformer, a variant of order-2 tensor transformer that requires $\mathcal{O}(|E|^2)$ memory. We also show that the model can be implemented efficiently with sparse tensor operations.

Approach We consider a graph represented as $G = (\mathbb{V}, E, \mathbf{A})$ with node set $\mathbb{V}$, set of edges E where $e_{ij} = (v_i, v_j) \in E$ denotes an edge pointing from v_i to v_j, and an feature tensor $\mathbf{A} \in \mathbb{R}^{n^2\times d}$ where $\mathbf{A}_{ijk} = 0 \forall k$ if $(v_i, v_j) \notin E$. We denote all self-loops in a graph as $\mathcal{S}(E) = \{e_{kk} \in E\}$. We denote in-neighbor of a node as $\mathcal{N}_{\text{in}}(v) = \{u \in \mathbb{V} | (u, v) \in E, u \neq v\}$, and out-neighbors of a node as $\mathcal{N}_{\text{out}}(v) = \{u \in \mathbb{V} | (v, u) \in E, u \neq v\}$. To define sparse tensor transformer layers, we first write the operations of dense encoder and decoder layers presented in main text in indexed format, and derive their sparse versions based on the assumption that topology (neighborhood) structure is maintained. For decoder layers that use cross-attention, we will derive the case where both inputs (query and key/value) are sparse because reducing either of the inputs to a dense tensor is straightforward.

Mixed-Order $(2 \to 1)$ Given dense $\mathbf{A} \in \mathbb{R}^{n^2 \times d}$, $\text{SelfAtt}_{2\to1} : \mathbb{R}^{n^2\times d} \to \mathbb{R}^{n\times d'}$ computes the following:

$$\begin{aligned}\text{SelfAtt}_{2\to1}(\mathbf{A}) = \text{diagonal}(\mathbf{A})W^1 + \text{diagonal}(\alpha^{(1,1)}\mathbf{A}^\top)W^2 + \text{diagonal}(\alpha^{(1,1)}\mathbf{A})W^3 \\ + \alpha^{(1,1)}\text{diagonal}(\mathbf{A})W^4 + \alpha^{(1,2)}\mathbf{A}W^5.\end{aligned} \tag{46}$$

where $W^1, ..., W^5 \in \mathbb{R}^{d\times d'}$ are learnable parameters. Index-wise, we can write the output as follows:

$$\begin{aligned}\text{SelfAtt}_{2\to1}(\mathbf{A})_i = \mathbf{A}_{ii}W^1 + \sum_{j\in\mathbb{V}\setminus\{i\}} \alpha^{(1,1)}_{ij}\mathbf{A}_{ij}W^2 + \sum_{j\in\mathbb{V}\setminus\{i\}} \alpha^{(1,1)}_{ij}\mathbf{A}_{ji}W^3 \\ + \sum_{j\in\mathbb{V}\setminus\{i\}} \alpha^{(1,1)}_{ij}\mathbf{A}_{jj}W^4 + \sum_{j,k\in\mathbb{V}\setminus\{i\}} 1_{j\neq k}\alpha^{(1,2)}_{ijk}\mathbf{A}_{jk}W^5,\end{aligned} \tag{47}$$

Now, assuming that input is sparse with topology given as E, we can reduce the computation as follows:

$$\begin{aligned}\text{SelfAtt}_{2\to1}(\mathbf{A})_i = \mathbf{A}_{ii}W^1 + \sum_{v_j\in\mathcal{N}_{\text{out}}(v_i)} \alpha^{(1,1)}_{ij}\mathbf{A}_{ij}W^2 + \sum_{v_j\in\mathcal{N}_{\text{in}}(v_i)} \alpha^{(1,1)}_{ij}\mathbf{A}_{ji}W^3 \\ + \sum_{e_{jj}\in\mathcal{S}(E)\setminus\{e_{ii}\}} \alpha^{(1,1)}_{ij}\mathbf{A}_{jj}W^4 + \sum_{e_{jk}\in E\setminus\mathcal{S}(E)\setminus\{e_{\cdot i}\in E\}\setminus\{e_{i\cdot}\in E\}} \alpha^{(1,2)}_{ijk}\mathbf{A}_{jk}W^5,\end{aligned} \tag{48}$$

With sparse matrix multiplication, above can be implemented with memory complexity of $\mathcal{O}(n|E|)$.

Mixed-Order $(2 \to 2)$ Given dense $\mathbf{A} \in \mathbb{R}^{n^2 \times d}$, $\text{SelfAtt}_{2\to2} : \mathbb{R}^{n^2\times d} \to \mathbb{R}^{n^2\times d'}$ computes the following:

$$\begin{aligned}\text{SelfAtt}_{2\to2}(\mathbf{A}) = \mathbf{I}\odot\mathbf{A}W^1 + (\mathbf{I}-\mathbf{1}\mathbf{1}^\top)\odot\mathbf{A}W^2 + (\mathbf{I}-\mathbf{1}\mathbf{1}^\top)\odot\mathbf{A}^\top W^3 \\ + (\mathbf{I}-\mathbf{1}\mathbf{1}^\top)\odot(\text{diagonal}(\mathbf{A})\mathbf{1}^\top)^\top W^4 + (\mathbf{I}-\mathbf{1}\mathbf{1}^\top)\odot(\text{diagonal}(\mathbf{A})\mathbf{1}^\top)W^5 \\ + \mathbf{I}\odot(\alpha^{(1,1)}\mathbf{A})W^6 + \mathbf{I}\odot(\alpha^{(1,1)}\mathbf{A}^\top)W^7 \\ + \text{diagonal}_{2,3}(\alpha^{(2,1)}\mathbf{A})W^8 + \text{diagonal}_{1,3}(\alpha^{(2,1)}\mathbf{A}^\top)W^9 \\ + \text{diagonal}_{2,3}(\alpha^{(2,1)}\mathbf{A}^\top)W^{10} + \text{diagonal}_{1,3}(\alpha^{(2,1)}\mathbf{A})W^{11} \\ + \text{diag}(\alpha^{(1,1)}\text{diagonal}(\mathbf{A}))W^{12} + \text{diag}(\alpha^{(1,2)}\mathbf{A})W^{13} \\ + \alpha^{(2,1)}\text{diagonal}(\mathbf{A})W_{14} + \alpha^{(2,2)}\mathbf{A}W_{15}.\end{aligned} \tag{49}$$

where $W^1, ..., W^{15} \in \mathbb{R}^{d\times d'}$ are learnable parameters. Index-wise, we can write the output as follows:

$$\begin{aligned}\text{SelfAtt}_{2\to2}(\mathbf{A})_{ij} = 1_{i=j}\mathbf{A}_{ii}W^1 + 1_{i\neq j}\mathbf{A}_{ij}W^2 + 1_{i\neq j}\mathbf{A}_{ji}W^3 + 1_{i\neq j}\mathbf{A}_{jj}W^4 + 1_{i\neq j}\mathbf{A}_{ii}W^5 \\ + 1_{i=j}\sum_{k\in\mathbb{V}\setminus\{j\}} \alpha^{(1,1)}_{ik}\mathbf{A}_{kj}W^6 + 1_{i=j}\sum_{k\in\mathbb{V}\setminus\{i\}} \alpha^{(1,1)}_{ik}\mathbf{A}_{ik}W^7 \\ + 1_{i\neq j}\sum_{k\in\mathbb{V}\setminus\{i,j\}} \alpha^{(2,1)}_{ijk}\mathbf{A}_{kj}W^8 + 1_{i\neq j}\sum_{k\in\mathbb{V}\setminus\{i,j\}} \alpha^{(2,1)}_{ijk}\mathbf{A}_{ik}W^9 \\ + 1_{i\neq j}\sum_{k\in\mathbb{V}\setminus\{i,j\}} \alpha^{(2,1)}_{ijk}\mathbf{A}_{jk}W^{10} + 1_{i\neq j}\sum_{k\in\mathbb{V}\setminus\{i,j\}} \alpha^{(2,1)}_{ijk}\mathbf{A}_{ki}W^{11} \\ + 1_{i=j}\sum_{k\in\mathbb{V}\setminus\{i\}} \alpha^{(1,1)}_{ik}\mathbf{A}_{kk}W^{12} + 1_{i=j}\sum_{k,l\in\mathbb{V}\setminus\{i\}} \alpha^{(1,2)}_{ikl}1_{k\neq l}\mathbf{A}_{kl}W^{13} \\ + 1_{i\neq j}\sum_{k\in\mathbb{V}\setminus\{i,j\}} \alpha^{(2,1)}_{ijk}\mathbf{A}_{kk}W^{14} + 1_{i\neq j}\sum_{k,l\in\mathbb{V}\setminus\{i,j\}} \alpha^{(2,2)}_{ijkl}1_{k\neq l}\mathbf{A}_{kl}W^{15},\end{aligned} \tag{50}$$

Now, assuming sparse input and output with topology given as E, we can reduce the computation as follows:

$$
\begin{aligned}
\text{SelfAtt}_{2\to 2}(\mathbf{A})_{ij;(i,j)\in E\setminus\mathcal{S}(E)} &= \mathbf{A}_{ij}W^2 + \mathbf{A}_{ji}W^3 + \mathbf{A}_{jj}W^4 + \mathbf{A}_{ii}W^5 \\
&+ \sum_{v_k\in\mathcal{N}_{\text{in}}(v_j)\setminus\{v_i\}} \alpha_{ijk}^{(2,1)}\mathbf{A}_{kj}W^8 + \sum_{v_k\in\mathcal{N}_{\text{out}}(v_i)\setminus\{v_j\}} \alpha_{ijk}^{(2,1)}\mathbf{A}_{ik}W^9 \\
&+ \sum_{v_k\in\mathcal{N}_{\text{out}}(v_j)\setminus\{v_i\}} \alpha_{ijk}^{(2,1)}\mathbf{A}_{jk}W^{10} + \sum_{v_k\in\mathcal{N}_{\text{in}}(v_i)\setminus\{v_j\}} \alpha_{ijk}^{(2,1)}\mathbf{A}_{ki}W^{11} \\
&+ \sum_{e_{kk}\in\mathcal{S}(E)\setminus\{e_{ii},e_{jj}\}} \alpha_{ijk}^{(2,1)}\mathbf{A}_{kk}W^{14} \\
&+ \sum_{e_{kl}\in E\setminus\mathcal{S}(E)\setminus\{e_{\cdot i}\in E\}\setminus\{e_{i\cdot}\in E\}\setminus\{e_{\cdot j}\in E\}\setminus\{e_{j\cdot}\in E\}} \alpha_{ijkl}^{(2,2)}\mathbf{A}_{kl}W^{15}, \\
\text{SelfAtt}_{2\to 2}(\mathbf{A})_{ii;(i,i)\in\mathcal{S}(E)} &= \mathbf{A}_{ii}W^1 + \sum_{v_k\in\mathcal{N}_{\text{in}}(v_i)} \alpha_{ik}^{(1,1)}\mathbf{A}_{ki}W^6 + \sum_{v_k\in\mathcal{N}_{\text{out}}(v_i)} \alpha_{ik}^{(1,1)}\mathbf{A}_{ik}W^7 \\
&+ \sum_{e_{kk}\in\mathcal{S}(E)\setminus\{e_{ii}\}} \alpha_{ik}^{(1,1)}\mathbf{A}_{kk}W^{12} + \sum_{e_{kl}\in E\setminus\mathcal{S}(E)\setminus\{e_{\cdot i}\in E\}\setminus\{e_{i\cdot}\in E\}} \alpha_{ikl}^{(1,2)}\mathbf{A}_{kl}W^{13},
\end{aligned}
\tag{51}
$$

With sparse matrix multiplication, above can be implemented with memory complexity of $\mathcal{O}(|E|^2)$.