On the Theoretical Expressive Power and the Design Space of Higher-Order Graph Transformers

Cai Zhou†[1] **Rose Yu**[*] **Yusu Wang**[*]
†Tsinghua University [*]University of California, San Diego
zhouc20@mails.tsinghua.edu.cn, {roseyu,yusuwang}@ucsd.edu

Abstract

Graph transformers have recently received significant attention in graph learning, partly due to their ability to capture more global interaction via self-attention. Nevertheless, while higher-order graph neural networks have been reasonably well studied, the exploration of extending graph transformers to higher-order variants is just starting. Both theoretical understanding and empirical results are limited. In this paper, we provide a systematic study of the theoretical expressive power of order-k graph transformers and sparse variants. We first show that, an order-k graph transformer without additional structural information is less expressive than the k-Weisfeiler Lehman (k-WL) test despite its high computational cost. We then explore strategies to both sparsify and enhance the higher-order graph transformers, aiming to improve both their efficiency and expressiveness. Indeed, sparsification based on neighborhood information can enhance the expressive power, as it provides additional information about input graph structures. In particular, we show that a natural neighborhood-based sparse order-k transformer model is not only computationally efficient, but also expressive – as expressive as k-WL test. We further study several other sparse graph attention models that are computationally efficient and provide their expressiveness analysis. Finally, we provide experimental results to show the effectiveness of the different sparsification strategies.

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. [1]Work done as an intern at UCSD.

1 INTRODUCTION

Recent years have witnessed great success in using the Transformer architecture for various applications, e.g., in natural language processing (Vaswani et al., 2017), computer vision (Dosovitskiy et al., 2020), and more recently graph learning (Rampasek et al., 2022; Bo et al., 2023). However, applying transformers to graphs is fundamentally different from texts or images because the graph topology and structure information cannot be easily captured by standard transformers. Furthermore, while graph neural networks benefit from higher-order extensions known as Invariant Graph Networks (IGN) (Maron et al., 2018; Geerts, 2020a), higher-order transformers have not yet been well studied, whose theoretical benefits and empirical strengths are still largely unexplored.

Specifically, on the theoretical front, analysis of the expressive power of graph transformers is currently limited, especially for the higher-order variants. Cai et al. (2023) show that a Message Passing Neural Network (MPNN) with a virtual node can approximate certain kernelized graph transformers (Performer (Choromanski et al., 2020) and Linear Transformer (Katharopoulos et al., 2020)). Kim et al. (2021) proposed a higher-order attention mechanism operating on k-tuples (instead of graph nodes), inspired by a generalization of linear equivariant layers (Maron et al., 2018). Kim et al. (2022) introduced a high-order graph transformer model called TokenGT, and showed that order-k TokenGT is as expressive as the so-called k-WL (order-k Weisfeiler-Lehman (Cai et al., 1989)) test. However, their result requires that the input k-tuples be equipped with orthonormal vectors (of length $O(n)$) as node identifiers. Furthermore, it is not clear how to choose such node identifiers in a permutation-invariant manner.

On the empirical front, k-transformers are computationally expensive, and the order-k transformer of (Kim et al., 2022) takes time $O(n^{2k})$ for a graph with n nodes. (Note that higher-order graph networks are also expensive; for example, the expressive order-k Invariant

Graph Network (i.e. k -IGN) also takes $O(n^{2k})$ time to compute for each layer.) Hence, it is valuable to explore sparse versions for improved efficiency.

Our Work. In this paper, we provide a systematic study of the theoretical expressive power of order-k graph transformers and their sparse variants. Our main contributions are as follows. See also the summary of some theoretical results in Table 1.

- In Section 3, after formally introducing a natural formulation of order-k transformers $\mathcal{A}_k$, we show that without "indices" information of k-tuples, $\mathcal{A}_k$ is strictly less expressive than k-WL. But when augmented with the indices information, its expressive power is at least that of k-WL. Note that however, each layer of $\mathcal{A}_k$ takes $O(n^{2k}d)$ time (see Table 1) where d is the network width (latent dimension). Unfortunately, similar to (Kim et al., 2022), the resulting model may not be invariant to the choices of the indices.
- In Section 4, we explore strategies to improve the efficiency of higher-order graph transformers while maintaining strong expressive power. In fact, sparsifying attention using the graph structure can enhance the expressive power. We propose several sparse high-order transformers in Section 4, and analyze their expressiveness and time complexity. A particularly interesting one is what we call the *neighbor-attention* mechanism, which, as shown in Table 1, significantly improves the time complexity (from $O(n^{2k})$ to $O(n^k)$ roughly speaking), while is as expressive as k-WL. Furthermore, note that this model doesn't use indices, and is permutation invariant. In Appendix C, we also study simplicial complexes-based higher-order transformers.
- In Section 5 and Appendix E, we also provide experimental results to show the relative performances of these higher-order transformers (although mostly of order 2), and compare them with SOTA methods in graph learning.

2 PRELIMINARIES AND BACKGROUND

In this section, we briefly review two notions: the order k-Weisfeiler-Lehman (k-WL) test, which is commonly used as a way to measure expressiveness of graph neural networks, as well as the k-IGN (Maron et al., 2018), which is the most expressive graph networks of a given order(k-IGN is as expressive as k-WL).

k-WL test is a procedure to test whether two input attributed graphs are potentially isomorphic or not. In the literature on graph networks, the k-WL procedure usually refers to the following iterative color-assignment scheme for an input graph G. The k-WL test will return that two graphs are "not isomorphic" if the collections of colors assigned at any iteration differ; otherwise, it will return "potentially isomorphic".

In particular, given a graph G, the k-WL procedure assigns colors to all k-tuples of $V(G)$ and iteratively updates them. The initial color $c_k^0(\mathbf{v}, G)$ of the tuple $\mathbf{v} \in V(G)^k$ is determined by the isomorphism type of the tuple $\mathbf{v}$ (Maron et al., 2019). At the t-th iteration, the color updating scheme is

$$c_k^t(\mathbf{v}, G) = \text{Hash}\Big(c_k^{t-1}(\mathbf{v}, G), \big(\{\{c_k^{t-1}(\psi_i(\mathbf{v}, u), G) | u \in V(G)\}\} | i \in [k]\big)\Big) \quad (1)$$

where $\psi_i(\mathbf{v}, u)$ means replacing the i-th element in $\mathbf{v}$ with u. The color of the entire graph is the multiset of all tuple colors,

$$c_k^t(G) = \text{Hash}\big(\{\{c_k^t(\mathbf{v}, G) | \mathbf{v} \in V(G)^k\}\}\big) \quad (2)$$

As mentioned earlier, two graphs are considered "potentially isomorphic" by k-WL if they have identical tuple color multisets for all iterations ts. It is known that 1-WL is equivalent to 2-WL in terms of differentiating graphs, while k-WL is strictly less powerful than $(k+1)$-WL for $k \geq 2$ (Cai et al., 1989).

k-IGN was introduced by Maron et al. (2018). An order k Invariant Graph Network (k-IGN) is defined as a function $F_k : \mathbb{R}^{n^k \times d_o} \to \mathbb{R}$ of the following form:

$$F_k = \text{MLP} \circ L_{k\to 0} \circ L_{k\to k}^{(T)} \circ \sigma \circ \ldots \circ \sigma \circ L_{k\to k}^{(1)}, \quad (3)$$

where each $L_{k\to k}^{(t)}$ is a (permutation) *equivariant linear layer* $\mathbb{R}^{n^k \times d_{t-1}} \to \mathbb{R}^{n^k \times d_t}$, $L_{k\to 0}$ is an *invariant linear layer* $\mathbb{R}^{n^k \times d_T} \to \mathbb{R}$, while σ is nonlinear activation function such as ReLU. In particular, it turns out that there exist $\text{bell}(k+l)$ number of basis tensors $\mathbf{B}^\mu$ and $\text{bell}(l)$ number of basis tensors $\mathbf{C}^\lambda$, such that any equivariant linear layer $L_{k\to l} : \mathbb{R}^{n^k \times d} \to \mathbb{R}^{n^l \times d'}$ from a k-order tensor to a l-order tensor can be written as follows: for order k input $\boldsymbol{X} \in \mathbb{R}^{n^k \times d}$,

$$L_{k\to l}(\boldsymbol{X})_{\boldsymbol{i}} = \sum_\mu \sum_{\boldsymbol{j}} \mathbf{B}_{\boldsymbol{i},\boldsymbol{j}}^\mu \boldsymbol{X}_{\boldsymbol{j}} w_\mu + \sum_\lambda \mathbf{C}_{\boldsymbol{i}}^\lambda b_\lambda. \quad (4)$$

Here $\boldsymbol{i} \in [n]^l, \boldsymbol{j} \in [n]^k$ are multi-indices, $w_\mu \in \mathbb{R}^{d\times d'}, b_\lambda \in \mathbb{R}^{d'}$ are weight and bias parameters, and $\mathbf{B}^\mu \in \mathbb{R}^{n^{l+k}}$. Note that an invariant layer is simply a special case of an equivariant layer $L_{k\to l}$ when $l = 0$. It is known that k-IGNs are as expressive as k-WL Maron et al. (2019); Geerts (2020a); Azizian and Lelarge (2020). Analogous to the construction of k-IGN, all graph networks including graph transformers have permutation equivariant intermediate layers, and permutation invariant final graph-level outputs.

Table 1: Summary of expressiveness and complexity of high-order transformers $\mathcal{A}_{k_1,k_2}$, their sparse variants, and simplicial transformer $\mathcal{AS}_{k_1:k_2}$. We also include the results of k-IGN of (Maron et al., 2018) for reference. Here n is the number of nodes; $\bar{D}$ is the average node degree; d is the latent dimension of model (sometimes omitted in main text); $\mathcal{S}_k$ is the set of k-simplices; *simplex neighbor* includes coboundaries, boundaries, upper and lower adjacent neighbors, while $\bar{D}_{\mathcal{S}}$ is the average number of these extended neighbors. Details of the simplicial transformers $\mathcal{AS}_*$s are in Appendix C.

Base model	Type	Computation reduction	Sparse attention	Enhancement	Complexity	Expressive power
k-IGN (Maron et al., 2018)	-	-	-	-	$O(n^{2k}d)$	$=k$-WL
$\mathcal{A}_k$	dense	-	-	-	$O(n^{2k}d)$	$\prec k$-WL
$\mathcal{A}_k$	dense	-	-	input indices	$O(n^{2k}d)$	$\succeq k$-WL(with ReLU)
$\mathcal{A}_k$	dense	kernelization	-	-	$O(n^k d^2)$	$\prec k$-WL
$\mathcal{A}_k^{\mathsf{Ngbh}}$	sparse	-	neighbor	-	$O(n^{k+1}kd)$	$=k$-WL
$\mathcal{A}_k^{\mathsf{LN}}$	sparse	-	local neighbor	-	$O(n^k k\bar{D}d)$	$\succeq \delta$-k-LWL
$\mathcal{A}_k^{\mathsf{VT}}$	sparse	-	virtual tuple	-	$O(n^k d)$	$\simeq$ kernelized $\mathcal{A}_k$
$\mathcal{AS}_k$	dense	-	-	$\mathbf{L}_k$	$O(\lvert\mathcal{S}_k\rvert^2 d)$	-
$\mathcal{AS}_{0:K}$	dense	-	-	$\mathcal{L}_{0:K}$	$O((\sum_{k=0}^{K}\lvert\mathcal{S}_k\rvert)^2 d)$	$\succeq$MPSN
$\mathcal{AS}_{0:K}^{\mathsf{SN}}$	sparse	-	simplex neighbor	-	$O((\sum_{k=0}^{K}\lvert\mathcal{S}_k\rvert)\bar{D}_{\mathcal{S}}d)$	=MPSN
$\mathcal{AS}_{0:K}^{\mathsf{VS}}$	sparse	-	virtual simplex	-	$O((\sum_{k=0}^{K}\lvert\mathcal{S}_k\rvert)d)$	$\simeq$ kernelized $\mathcal{AS}_{0:K}$

3 THEORETICAL ANALYSIS OF HIGH-ORDER TRANSFORMERS

In this section, we first introduce a natural notion of higher-order transformers, which is slightly more general than the one in Kim et al. (2022). We show that a plain order-k transformer without additional structural information is *strictly less expressive* than k-WL. On the other hand, adding explicit tuple indices as part of the input tuple features leads to a k-transformer that is at least as powerful as k-WL. However, it may also break the permutation invariance (not invariant to the choice of the indices). The proofs and some additional results can be found in Appendix B.1.

Definition 3.1 (Order k_1, k_2-Transformer Layer)**.** A *(cross-attention) transformer layer* $\mathcal{A}_{k_1,k_2} : \mathbb{R}^{n^{k_1}\times d} \times \mathbb{R}^{n^{k_2}\times d} \to \mathbb{R}^{n^{k_1}\times d'}$ takes a query tensor $\boldsymbol{X} \in \mathbb{R}^{n^{k_1}\times d}$ and a key tensor $\boldsymbol{Y} \in \mathbb{R}^{n^{k_2}\times d}$ as input, and is defined as

$$\mathcal{A}_{k_1,k_2}(\boldsymbol{X},\boldsymbol{Y}) = \mathrm{softmax}\Big(\underbrace{(\boldsymbol{X}Q)}_{\in\mathbb{R}^{n^{k_1}\times d_k}} \underbrace{(\boldsymbol{Y}K)^\top}_{\in\mathbb{R}^{d_k\times n^{k_2}}} \Big) \underbrace{(\boldsymbol{Y}V)}_{\in\mathbb{R}^{n^{k_2}\times d'}} . \tag{5}$$

Here $Q \in \mathbb{R}^{d\times d_k}$, $K \in \mathbb{R}^{d\times d_k}$, $V \in \mathbb{R}^{d\times d'}$ are the learnable weight matrices where d_k is the latent dimension, and the softmax is performed row-wise on an $n^{k_1} \times n^{k_2}$ matrix. In practice d, d_k, d' are usually of the same magnitude and are thus both denoted as $O(d)$ regarding complexity. We may also sometimes omit the $O(d)$ factor in the main text when we care more about the complexity w.r.t number of nodes n.

A *self-attention transformer layer* is when $\boldsymbol{X} = \boldsymbol{Y}$, $k_1 = k_2 = k$, which we also refer to as *(order) k-transformer layer*, and denoted by $\mathcal{A}_k$ for simplicity. Throughout the paper, we mostly talk about self-attention transformers (which is standard in the literature). However, some experiments using cross-attention transformers are given in Appendix E.

A k-transformer is composed of many layers of the above k-transformer layers. Note that the above definition is for single-head attention, but it is easy to extend it to multi-head attention analogous to standard transformers. Also, following the standard setting, we allow bias terms after multiplying $\boldsymbol{X}$ with query, key and value weight matrices; although these are omitted in the above definition for clarity. Furthermore, residual connection and input/output MLPs are always allowed between these k-transformer layers. Note that Definition 3.1 is generic and not limited to graph data. For graph input, similar to the k-WL procedure, the feature initialization of k-tuples can be based on the isomorphism types of the tuples (Maron et al., 2019).

Theoretical Expressive Power of Order-k Transformer. Next, we investigate the expressive power of order-k transformer in terms of the k-WL hierarchy. A subtle issue here is the use of indices: k-WL test assumes that the graph nodes are indexed, and a k-tuple is explicitly **indexed** by k indices of those nodes in this tuple. This index is only used to compute the "neighbors" of a specific k-tuple (see the use of $\psi_i(\boldsymbol{v}, u)$ in Eqn (1). The entire procedure is still *independent* to the indexing. Such structural information unfortunately is lost in a k-transformer layer, where the self-attention mechanism cannot differentiate such "neighbors". It is therefore not surprising that we have the following negative result (proof in Appendix B.1.2).

Theorem 3.2. *Without taking tuple indices as inputs, $\mathcal{A}_k$ is strictly less expressive than k-WL.*

The limited expressive power of $\mathcal{A}_k$ is not desirable: It suffers from high computational cost – indeed, the complexity for one $\mathcal{A}_k$ layer is the same as k-IGN, which is $O(n^{2k})$. Furthermore, it is less expressive than k-IGN (which is as expressive as k-WL).

A natural step forward is to make $\mathcal{A}_k$ more expressive through structural enhancements. Given the discussion of the use of "indices" in k-WL earlier, a natural strategy is to bring in structural information by augmenting the inputs with tuple indices. Indeed, as Theorem 3.3 below shows, this enhancement improves the expressiveness to the same as k-WL. Here we take the k-dimensional tuple indices as the model inputs, where each index $\boldsymbol{i} \in [n]^k$ is the multi-index of the same definition as in k-IGN (Maron et al., 2018).

Theorem 3.3. *For inputs $\boldsymbol{X} \in \mathbb{R}^{n^k \times (d+k)}$ where each element is a concatenation of a d-dimensional tuple feature and k-dimensional index of the tuple, one layer of $\mathcal{A}_k$ with latent dimension $O(k)$ and k heads augmented with input MLPs, residual connection feed-forward layers can approximate one k-WL iteration arbitrarily well. If the softmax function is replaced by element-wise ReLU activation, $\mathcal{A}_k$ can exactly simulate k-WL.*

Here we provide some brief comments on Theorem 3.3. It is interesting to see that $\mathcal{A}_k$ with ReLU activation augmented by tuple indices can simulate k -WL and thus is permutation invariant. Unfortunately, unlike k-IGN, the resulting model is not guaranteed to be permutation invariant to choices of tuple indices - for example, the output of the softmax version of $\mathcal{A}_k$ is not permutation invariant to the tuple indices. Hence the above theorem is only of theoretical interest. Note that the same issue also applies to TokenGT (Kim et al., 2022) – in fact, they assume that each node has a distinct orthonormal vector as a "node identifier", which is an even stronger requirement than using just indices. Each node identifier needs a size of $O(n)$, thus TokenGT requires more time complexity ($O(n^{2k+1})$) to achieve k-WL expressiveness than our construction ($O(n^{2k})$). Our proof is also different from that of Kim et al. (2022). Instead of approximating the equivalence class basis in k-IGN as they do, we directly simulate k-WL. See Appendix B.1.2 for the proof of the above theorem, as well as more discussions regarding our advantages over the result of Kim et al. (2022).

4 EFFICIENT AND EXPRESSIVE HIGH-ORDER GRAPH TRANSFORMERS

The plain order-k graph transformer model $\mathcal{A}_k$ suffers from $O(n^{2k})$ time complexity, limited expressive power (Theorem 3.2) or broken permutation invariance (Theorem 3.3). In this section, we explore sparse high-order graph transformers to address these issues. In particular, in Section 4.1, we study a more general kernelization strategy to sparsify self-attentions. The more interesting exploration is presented in Section 4.2, where we inject graph structure inductive biases to improve efficiency, while maintaining or even improving the expressive power of the resulting model. In Section 4.3, we also discuss how to reduce the computational complexity by using a reduced set of k-tuples with most details in Appendix C. The proofs of all the theorems in this section are in Appendix B.2.

Contrary to the explicit use of indexing as in Theorem 3.3 (as well as the use of orthonormal node identifiers for TokenGT Kim et al. (2022)), all the models in this section do not require *explicit indices as part of input features*, and the resulting attention layer is *permutation equivariant*.

4.1 Kernelized Attention

Kernelized attention has been commonly used for (standard) transformers (Choromanski et al., 2020; Katharopoulos et al., 2020). In this subsection, we generalize kernelized attention to higher-order transformers, reducing the complexity from $O(n^{2k}d)$ to $O(n^k d^2)$, where d is the feature dimension. We also provide a theoretical analysis of such generalization.

Such a generalization is rather straightforward. This is because the order-k self-attention layer essentially has the same structure as a standard self-attention layer, where the only difference is that the input tokens are now those k-tuples (instead of nodes in the standard transformer). For completeness, we include the formulation here. Specifically, a single head order-k self-attention layer can be re-formulated as

$$\boldsymbol{X}_i = \sum_{j=1}^{n^k} \frac{\kappa(\boldsymbol{X}_i Q, \boldsymbol{X}_j K)}{\sum_{l=1}^{n^k} \kappa(\boldsymbol{X}_i Q, \boldsymbol{X}_l K)} \cdot (\boldsymbol{X}_j V) \tag{6}$$

where $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the softmax kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(\mathbf{x}^\top \mathbf{y})$. The kernel trick approximates the softmax via $\kappa(\mathbf{x}, \mathbf{y}) =< \Phi(\mathbf{x}), \Phi(\mathbf{y}) >\approx \phi(\mathbf{x})^T \phi(\mathbf{y})$ where the first equation is by Mercer's theorem and the latter is a low-dimensional approximation with transformation $\phi : \mathbb{R}^d \to \mathbb{R}^m$. In Performer (Choromanski et al., 2020), $\phi(\mathbf{x}) = \frac{\exp\left(-\frac{||\mathbf{x}||_2^2}{2}\right)}{\sqrt{m}} [\exp(\mathbf{w}_1^\top \mathbf{x}), \ldots, \exp(\mathbf{w}_m^\top \mathbf{x})]$ where $\mathbf{w}_k \sim \mathcal{N}(0, I_d)$. In Linear Transformer (Katharopoulos et al., 2020), $\phi(\mathbf{x}) = elu(\mathbf{x}) + 1$. Then we can further rewrite the attention as

$$\boldsymbol{X}_i = \frac{\left(\phi(\boldsymbol{X}_i Q)^\top \sum_{j=1}^{n^k} \left(\phi(\boldsymbol{X}_j K) \otimes (\boldsymbol{X}_j V)\right)\right)^\top}{\phi(\boldsymbol{X}_i Q)^\top \sum_{l=1}^{n^k} \phi(\boldsymbol{X}_l K)} \tag{7}$$

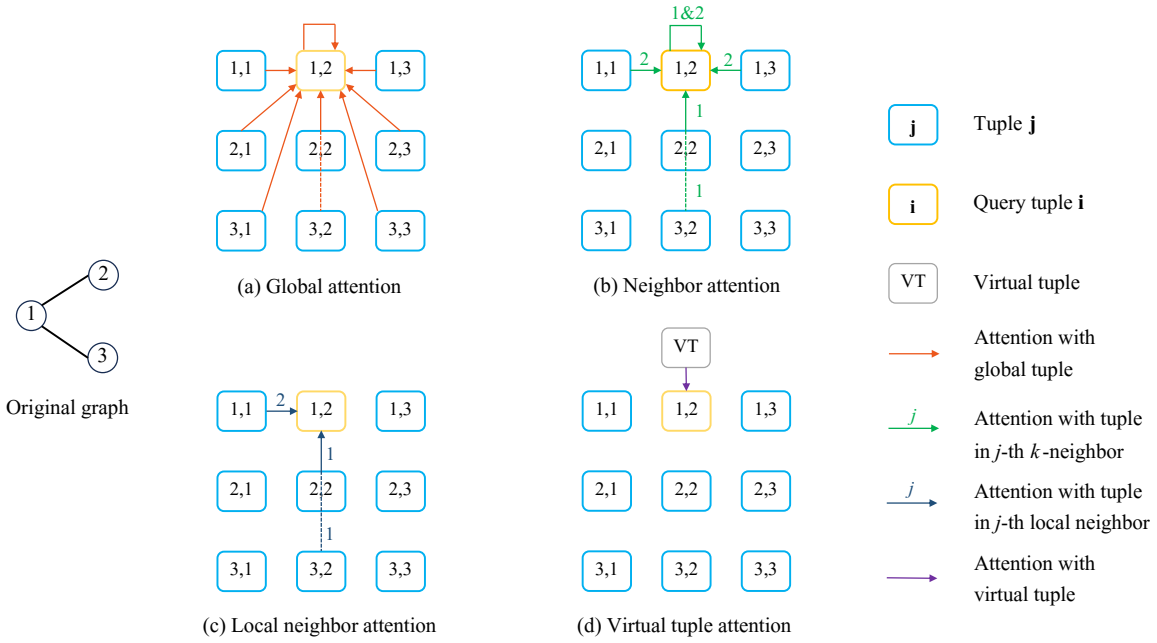


Figure 1: Variants of k-th order self-attention, i.e. $\mathcal{A}_k$ and its sparse forms. In the figure order $k = 2$, number of nodes $n = 3$. For simplicity, we only show the attention of query token $\boldsymbol{i} = (1, 2)$, and all n^k (real) tuples are calculated with the same rule. The dashed lines are only for aesthetic illustration. (a) Global attention (plain $\mathcal{A}_k$), the query token computes attention with all n^k tuples. (b) Neighbor attention, the query token computes attention with its k-neighbors; k-neighbor is of the same definition as in k-WL. (c) Local neighbor attention, where the query token computes attention with only its local neighbors; local neighbor is of the same definition as in (Morris et al., 2020). (d) Virtual tuple attention, the query token only computes attention with the virtual tuples (we only display one for simplicity), while each virtual tuple computes attention with all other real tuples.

where $\otimes$ is the outer product. Note that the two summations $\sum_{j=1}^{n^k} \left(\phi(\boldsymbol{X}_j K) \otimes (\boldsymbol{X}_j V)\right) \in \mathbb{R}^{md'}$ and $\sum_{l=1}^{n^k} \phi(\boldsymbol{X}_l K)$ are shared for all query tokens, while the former has the bottleneck complexity $O(n^k m d')$. Again, since m, d' are comparable to d in scale , the total time complexity of kernelized $\mathcal{A}_k$ is $O(n^k d^2)$. The following results show whether k-IGN and kernelized $\mathcal{A}_k$ can approximate each other.

Theorem 4.1. *k-IGN can approximate kernelized $\mathcal{A}_k$ with Linear Transformer or Performer architectures arbitrarily well. Kernelized $\mathcal{A}_k$ is strictly less powerful than k-IGN.*

4.2 Sparse Attention via Neighbor, Local Neighbor and Virtual Tuple

The aforementioned kernel tricks reduce model complexity from the perspective of computation and are applicable to general higher-order transformer. However, their attention mechanisms are still dense, in the sense that every query token aggregates information from all other tokens. Additionally, we haven't incorporated any inductive biases of graph and topology.

In this section, we aim to design sparse attention mechanisms that have the same or even stronger expressive power compared with full attention $\mathcal{A}_k$. This could be possible, because when each tuple only computes attention with local tuples (generalized 'neighbors' defined according to certain rules), the model becomes not only sparse in computation, but also aware of part of structural information, which could be regarded as some sort of enhancement.

In particular, we explore three types of sparsification strategies (see Figure 1), and study both their computational complexity and expressiveness. Let $[n]$ denote the set of integers from 1 to n. In all the constructions below, we will index all graph nodes (arbitrarily) from 1 to n, and index each k-tuple by $\boldsymbol{i} = (i_1, \ldots, i_k) \in [n]^k$. These indices will only be used in the selection of "neighbors", and the resulting attention layer will be permutation equivariant/invariant.

Neighbor Attention. The first variant is called *neighbor attention* mechanism. Similar to the k-WL procedure (recall Eqn (1)), given a k-tuple $\boldsymbol{i} \in [n]^k$, its j-th k-neighbor is the n number of k-tuples in the set

$\{\{\psi_j(\boldsymbol{i}, u) \mid u \in [n]\}\}$, where $\psi_j(\boldsymbol{i}, u)$ means replacing the j-th element in $\boldsymbol{i}$ with u. In the (multi-head) k-order neighbor attention, denoted as $\mathcal{A}_k^{\mathsf{Ngbh}}$, each query tuple $\boldsymbol{i}$ only computes its attention with its k number of k-neighbors (each consists of n number of k-tuples) in each head respectively and concatenates the results. Particularly, when the model has k heads,

$$\Big(\mathcal{A}_k^{\mathsf{Ngbh}}(\boldsymbol{X}, \boldsymbol{X})\Big)_{\boldsymbol{i}} = \text{Concat}\Big[\text{softmax}\Big((x_{\boldsymbol{i}}Q^j)^\top \big(x_{[\psi_j(\boldsymbol{i},u)|u\in[n]]}K^j\big)\Big)\big(x_{[\psi_j(\boldsymbol{i},u)|u\in[n]]}V^j \big| j \in [k]\big)\Big] \tag{8}$$

where $x_{[\psi_j(\boldsymbol{i},u)|u\in[n]]} \in \mathbb{R}^{n\times d},\ j \in [k]$ is just the feature of j-th neighbor of $\boldsymbol{i}$, and $Q^j \in \mathbb{R}^{d\times d_k}, K^j \in \mathbb{R}^{d\times d_k}, V^j \in \mathbb{R}^{d\times d'} (j \in [k])$ are weight matrices (parameters) of the j-th head. One may easily observe that this neighbor attention $\mathcal{A}_k^{\mathsf{Ngbh}}$ resembles k-WL algorithm, as they both update their representations using its k-neighbors. Now we show (proof in Appendix B.2.2) that neighbor attention $\mathcal{A}_k^{\mathsf{Ngbh}}$ is as powerful as k-WL, making it more expressive than $\mathcal{A}_k$ while enjoying much lower complexity.

Theorem 4.2. *Neighbor attention $\mathcal{A}_k^{\mathsf{Ngbh}}$ with residual connection, output MLPs, and k heads is as powerful as k-WL. Each such layer has $O(n^{k+1}kd)$ time complexity.*

Interestingly, while $\mathcal{A}_k^{\mathsf{Ngbh}}$ may assign different weights to the n tuples inside each k-neighbor via attentions, it has the same theoretical expressive power as k-IGN. But neighbor attention is much more efficient than k-IGN ($O(n^{k+1}kd)$ vs $O(n^{2k}d)$). Furthermore, we think that this more flexible attention might make neighbor attention better than k-IGN in real-world tasks, even regardless of their time complexity.

Local Neighbor Attention. Morris et al. (2020) proposed a family of δ-k-dimensional WL algorithms, which is strictly more powerful than k-WL. Denote by $\boldsymbol{i}_j$ the j th element in tuple $\boldsymbol{i}$ and $N(i)$ the neighbors of node i. Compared with the k-neighbor in k-WL, δ-k-WL augments each j-th k-neighbor with the connectivity between the node being replaced $\boldsymbol{i}_j$ and the n nodes replacing it. Formally, δ-k-WL computes

$$c^t_{\delta\text{-}k}(\boldsymbol{i}, G) = \text{Hash}\Big(c^{t-1}_{\delta\text{-}k}(\boldsymbol{i}, G), \Big(\{\{\big(c^{t-1}_{\delta\text{-}k}(\psi_j(\boldsymbol{i}, u), G), \text{adj}(\boldsymbol{i}_j, u)\big) | u \in V(G)\}\big| j \in [k]\Big)\Big) \tag{9}$$

where $\text{adj}(\boldsymbol{i}_j, u) = \mathbb{1}(u \in N(\boldsymbol{i}_j))$ is a boolean variable indicating whether node $\boldsymbol{i}_j$ and node u are connected. Using the idea of δ-k-WL, we can make $\mathcal{A}_k^{\mathsf{Ngbh}}$ potentially more expressive than k-WL by incorporating $\text{adj}(\boldsymbol{i}_j, u)$ via attention bias or attention reweighting, which we denote as $\mathcal{A}_k^{\mathsf{Ngbh}+}$, see Appendix B.2 for details.

However, despite being potentially more expressive, $\mathcal{A}_k^{\mathsf{Ngbh}+}$ has the same $O(n^{k+1}kd)$ complexity as $\mathcal{A}_k^{\mathsf{Ngbh}}$. Now we continue to present a more sparse attention variant, namely *local neighbor attention*. Recall Morris et al. (2020) proposed a local variant algorithm named δ-k-LWL, which only updates each tuple with its local neighbors. Specifically, the j-th *local neighbor* of a k-tuple $\boldsymbol{i}$ is defined as

$$\mathcal{N}_j^{Local}(\boldsymbol{i}) := \{\{\psi_j(\boldsymbol{i}, v) | v \in N(\boldsymbol{i}_j)\}\} \tag{10}$$

which is a multi-set consists of $D(\boldsymbol{i}_j)$ k-tuples where $D(\boldsymbol{i}_j)$ is the degree of node $\boldsymbol{i}_j$ - each tuple replacing element $\boldsymbol{i}_j$ with its neighboring nodes $N(\boldsymbol{i}_j)$ (instead of all n possible nodes as in k-WL). More detailed descriptions are given in Appendix B.2.2.

Inspired by Morris et al. (2020), we propose the (k-head) order-k *local neighbor attention* $\mathcal{A}_k^{\mathsf{LN}}$,

$$\Big(\mathcal{A}_k^{\mathsf{LN}}(\boldsymbol{X}, \boldsymbol{X})\Big)_{\boldsymbol{i}} = \text{Concat}\Big[\text{softmax}\Big((x_{\boldsymbol{i}}Q^j)^\top \big(x_{[\psi_j(\boldsymbol{i},u)|u\in N(\boldsymbol{i}_j)]}K^j\big)\Big)\big(x_{[\psi_j(\boldsymbol{i},u)|u\in N(\boldsymbol{i}_j)]}V^j \big| j \in [k]\big)\Big] \tag{11}$$

where $Q^j, K^j, V^j (j \in [k])$ are weight parameters of the j-th head. In other words, $\mathcal{A}_k^{\mathsf{LN}}$ updates each query tuple according to its local neighbors defined in Equation (10). For example, it is easy to see that the attention in Figure 1 (c) is sparser than Figure 1 (b); e.g., tuple $(1, 3)$ is in the 2-th 2-neighbor of query tuple $(1, 2)$, but not in the 2-th local neighbor of $(1, 2)$ since nodes 2 and 3 are disconnected.

Theorem 4.3. *Local neighbor attention $\mathcal{A}_k^{\mathsf{LN}}$ with residual connection, output MLPs, and k heads is at least as powerful as δ-k-LWL. Each such layer has $O(n^k k\bar{D}d)$ time complexity, where $\bar{D}$ is the average node degree.*

Virtual Tuple Attention. We now present the last model of sparse attention, *virtual tuple attention*. Virtual node is a widely adopted heuristic technique for message-passing neural networks (MPNNs), and more recently graph transformers (Shirzad et al., 2023). The approximation power and expressive power of MPNN + virtual node have been studied in Cai et al. (2023).

Similar to the idea of the virtual node, we introduce a virtual tuple and propose the virtual tuple attention $\mathcal{A}_k^{\mathsf{VT}}$, where the virtual tuple computes attention with all other real tuples. Each real tuple only computes attention with the virtual tuple (as there is only one key in this case, the softmax always outputs 1, thus similar to message passing). The time complexity of virtual tuple attention is $O(n^k d)$, since each of the n^k

real k-tuples only needs to compute attention with the virtual tuple. Note that in practice, we can use multiple virtual tuples to capture more complex patterns.

Mathematically, denote the feature of the virtual tuple as x', the input is augmented to $\boldsymbol{X}' = [\boldsymbol{X}, x'] \in \mathbb{R}^{(n^k+1)\times d}$, then $\mathcal{A}_k^{\mathsf{VT}}$ is calculated as

$$\mathcal{A}_k^{\mathsf{VT}}(\boldsymbol{X}', \boldsymbol{X}')_{n^k+1} = \mathrm{softmax}\Big((x'Q^1)^\top(\boldsymbol{X}K^1)\Big)\boldsymbol{X}V^1 \tag{12}$$

$$\mathcal{A}_k^{\mathsf{VT}}(\boldsymbol{X}', \boldsymbol{X}')_{\boldsymbol{i}} = x'V^2 \tag{13}$$

Note that as $\mathcal{A}_k$ does not include tuple indices nor equivalence class basis, its properties are the same as first-order transformer with n^k one-dimensional input tokens. Therefore, when allowed to be augmented with input and output MLPs, virtual tuple attention can be regarded as a natural extension of "simplified MPNN + virtual node" (Cai et al., 2023), and analysis of that "simplified MPNN + virtual node" can be applied directly to $\mathcal{A}_k^{\mathsf{VT}}$. See Appendix B.2 for more precise descriptions and details. Following the results of (Cai et al., 2023), we thus obtain:

Proposition 4.4. *$O(1)$ depth and $O(1)$ width virtual tuple attention $\mathcal{A}_k^{\mathsf{VT}}$ can approximate kernelized $\mathcal{A}_k$ with Performer or Linear-Transformer architecture arbitrarily well.*

Analogously, as $O(1)$ depth and $O(n^d)$ width MPNN + VN can simulate full standard transformer (Cai et al., 2023), the $O(1)$ depth and $O(n^{kd})$ width virtual tuple attention can thus simulate full $\mathcal{A}_k$ transformer.

4.3 Reducing Input k-tuples

The three sparse attention models in the previous section require $O(n^k)$ complexity which is dictated by the number of input tokens (i.e. the number of k-tuples). An orthogonal direction to reduce computational complexity is to reduce the number of query tokens.

Simplicial Attention. Instead of all k-tuples, one can select a subset of k-tuples according to certain systematic rules. Simplicial complexes provide a language to model such choices. For example, in applied and computational topology, one approach is to view an input graph as the 1-skeleton of a hidden space, and there have been various works to construct a simplicial complex from the graph that can reflect low or high-dimensional topological features of this hidden space (Dey and Wang, 2022). Note that a p-dimensional simplex is intuitively spanned by $p+1$ number of vertices. As an example, a simple way to construct a p-simplicial is to include all those $(p+1)$-tuples of graph nodes that form a clique in the input graph. (The resulting simplicial complex is the so-called flag complex, or clique complex.) In general, the number of $(k-1)$-simplices is much smaller than all possible k-tuples of graph nodes. In Appendix C, we propose **simplicial attention variants** and analyze their theoretical properties. To distinguish from the tuple-based transformers in our main text, we use $\mathcal{AS}$ to denote simplicial transformers both in Table 1 and in experimental results.

Random Sampling. Finally, we remark that one can also use a random subset of k-tuples (either uniformly, or w.r.t. some probabilistic distribution depending on input graph structure). We provide some empirical results of sampling connected 3-tuples in Appendix E. It will be an interesting future direction to explore how to obtain theoretical guarantees in expressiveness or approximation power under certain sampling strategies.

5 EXPERIMENTS

We conduct experiments on both synthetic datasets and real-world datasets. Using our sparse attention techniques, we can now scale order-2 graph transformers to datasets containing relatively large graphs, such as the long-range graph benchmark (LRGB) (Dwivedi et al., 2022). Our higher-order graph transformers and simplicial transformers show superior expressivity on synthetic datasets, and achieve competitive performance across several real-world datasets.

Due to limited space, we leave most experimental results, implementation details, and in-depth analysis in Appendix E. As representative results, we report the performance of our different higher-order transformer variants on (1) synthetic datasets with structure awareness tasks (Muller et al., 2023), including detecting edges and distinguishing non-isomorphic circular skip links (CSL) graphs, see Table 2; (2) Zinc12k (Dwivedi et al., 2020), a popular molecular property prediction dataset, see Table 3. See Appendix E for more results, including substructure counting for synthetic tasks, as well as OGB (Hu et al., 2020) and LRGB (Dwivedi et al., 2022) benchmarks for real-world tasks. Our main goal is to verify the theoretical properties or scalability of different models, and provide empirical analysis on their pros and cons.

Results on Synthetic Datasets. Table 2 shows the results on some synthetic datasets to study the capability of various models to capture graph "structures". Edge detection is a binary classification task to predict whether there is an edge connecting two given nodes, while the CSL test is a ten-way classification task to distinguish non-isomorphic circular skip links (CSL) graphs, which requires awareness of distance. We adopt the same experimental settings and baseline choices as (Muller et al., 2023). Edge detection is an easier task,

Table 2: Structure awareness tasks on synthetic datasets. Shown is the mean ± std of 5 runs with different random seeds. Perfect results are shown in **bold**. The experimental settings and baseline results are adopted from (Muller et al., 2023).

Model	Edge detection 2-way Accuracy ↑	CSL 10-way Accuracy ↑
GIN	98.11 ± 1.78	10.00 ± 0.00
Graphormer	97.67 ± 0.97	90.00 ± 0.00
Transformer	55.84 ± 0.32	10.00 ± 0.00
Transformer+LapPE	98.00 ± 1.03	**100.00** ± 0.00
Transformer+RWSE	97.11 ± 1.73	**100.00** ± 0.00
$\mathcal{A}_2$	**100.00** ± 0.00	10.00 ± 0.00
$\mathcal{A}_2$+LapPE	**100.00** ± 0.00	**100.00** ± 0.00
$\mathcal{A}_2$+RWSE	**100.00** ± 0.00	**100.00** ± 0.00
$\mathcal{A}_2^{\mathsf{LN}}$	**100.00** ± 0.00	**100.00** ± 0.00
$\mathcal{A}_2^{\mathsf{LN}}$+LapPE	**100.00** ± 0.00	**100.00** ± 0.00
$\mathcal{A}_2^{\mathsf{LN}}$+RWSE	**100.00** ± 0.00	**100.00** ± 0.00
$\mathcal{AS}_{0:1}$+attn.bias	97.80 ± 0.33	36.67 ± 0.00
$\mathcal{AS}_{0:1}$+attn.bias+LapPE	98.47 ± 0.11	**100.00** ± 0.00
$\mathcal{AS}_{0:1}$+attn.bias+RWSE	98.10 ± 0.25	**100.00** ± 0.00
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$	96.16 ± 0.37	20.00 ± 0.00
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$+LapPE	99.98 ± 0.01	**100.00** ± 0.00
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$+RWSE	99.54 ± 0.08	**100.00** ± 0.00

Table 3: Results on ZINC (Dwivedi et al., 2020). Shown is the mean ± std of 5 runs with different random seeds. Highlighted are the first, second and third results. Experimental settings and baseline results are adopted from (Rampasek et al., 2022).

Model	Test MAE ↓
GCN (Kipf and Welling, 2016)	0.367 ± 0.011
GAT (Velickovic et al., 2017)	0.384 ± 0.007
GatedGCN (Bresson and Laurent, 2017)	0.282 ± 0.015
PNA (Corso et al., 2020)	0.188 ± 0.004
CIN (Bodnar et al., 2021a)	0.079 ± 0.006
GIN-AK+ (Zhao et al., 2021)	0.080 ± 0.001
SAN (Kreuzer et al., 2021)	0.139 ± 0.006
Graphormer (Ying et al., 2021)	0.122 ± 0.006
EGT (Hussain et al., 2021)	0.108 ± 0.009
GPS (Rampasek et al., 2022)	0.070 ± 0.004
$\mathcal{A}_2$-Performer (ours)	0.155 ± 0.008
$\mathcal{A}_2^{\mathsf{VT}}$ (ours)	0.186 ± 0.009
$\mathcal{A}_2^{\mathsf{Ngbh}}$ (ours)	0.081 ± 0.005
$\mathcal{A}_2^{\mathsf{Ngbh+}}$ (ours)	0.075 ± 0.005
$\mathcal{A}_2^{\mathsf{LN}}$ (ours)	0.086 ± 0.006
$\mathcal{A}_2^{\mathsf{LN+VT}}$ (ours)	0.069 ± 0.005
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$ (ours)	0.080 ± 0.004
$\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ (ours)	0.073 ± 0.004

while CSL requires expressivity more powerful than 1-WL. Indeed, without position/structure encodings (PE/SE), we see that GIN, Transformer, and $\mathcal{A}_2$ fail in CSL task. Order-2 transformer with *local neighbor attention* $\mathcal{A}_2^{\mathsf{LN}}$ achieves perfect results in both tasks without any PE/SE, indicating that it sometimes can distinguish non-isomorphism graphs that 1-WL fails. This is consistent with our theory in Theorem 4.3, as it is at least as expressive as δ-2-LWL, which might differentiate graphs not distinguished by 1-WL (although it may also fail to distinguish graphs that can be differentiated by 1-WL). In general, these high-order models also benefit from PE and SE.

Results on Real-World Datasets. Zinc-12k (Dwivedi et al., 2020) is a popular real-world dataset containing 12k molecules. The task is a graph-level molecular property (constrained solubility) regression. We adopt the experimental settings and SOTA baseline results from (Rampasek et al., 2022). The results in Table 3 reveal that: (1) $\mathcal{A}_2$-Performer and $\mathcal{A}_2^{VT}$ achieve similar results, verifying Theorem 4.4; however, there are no theoretical guarantees in their expressive power, which aligns with the fact that their results are not highly competitive. (2) Both $\mathcal{A}_2$ and simplicial transformer $\mathcal{AS}_{0:1}$ achieve satisfactory results (< 0.09 test MAE) when using only (local) neighbors or simplex neighbors, revealing the importance of local structure awareness. (3) Although virtual tuple/simplex attention does not provably enhance expressivity, these empirical mechanisms can improve the practical performance of other sparse attentions: $\mathcal{A}_2^{\mathsf{LN+VT}}$ and $\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ achieve 0.069 and 0.073 test MAE respectively (better or comparable with SOTA GPS results (Rampasek et al., 2022), indicating the empirical strength of global information.

In addition, we also consider positional encoding for our high-order transformers. The approaches to add positional encoding to our high-order transformers are detailed in Appendix E.1. We verify the effectiveness of positional encoding to our models on Zinc and Alchemy (Chen et al., 2019), both of which are graph-level regression datasets to predict molecule properties. In both datasets, we consider positional encodings (PE), including SignNet/BasisNet Lim et al. (2022), SPE Huang et al. (2023) and MAP Ma et al. (2023a). Baseline results are adopted from these papers correspondingly.

As shown in Table 4 and Table 5, on both Zinc and Alchemy datasets, our high-order transformers can benefit from positional encoding, and achieve much more competitive performance compared with simple message-passing based GNNs.

In summary, our models are the first second-order attention-only methods that achieve results comparable to SOTA methods on Zinc (and also other real-world datasets in Appendix E). From the perspective

Table 4: More results on ZINC with positional encodings (PE). Shown is the mean ± std of 4 runs with different random seeds.

Model	PE	Test MAE ↓
GatedGCN	SignNet(8)	0.121 ± 0.005
GatedGCN	SignNet(All)	0.100 ± 0.007
GatedGCN	MAP(8)	0.120 ± 0.002
GINE	SignNet(16)	0.147 ± 0.005
GINE	SignNet(All)	0.102 ± 0.002
PNA	SignNet(8)	0.105 ± 0.007
PNA	SignNet(All)	0.084 ± 0.006
PNA	MAP(8)	0.101 ± 0.005
GIN	SPE(8)	0.074 ± 0.001
GIN	SPE(All)	0.069 ± 0.004
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$ (ours)	SignNet(8)	0.079 ± 0.005
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$ (ours)	SignNet(All)	0.078 ± 0.006
$\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ (ours)	SPE(8)	0.072 ± 0.005
$\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ (ours)	SPE(All)	0.067 ± 0.005

Table 5: Experiments on Alchemy Chen et al. (2019) with positional encodings (PE). Shown is the mean ± std of 4 runs with different random seeds.

Model	PE	Test MAE ↓
GIN	None	0.112 ± 0.001
GIN	SignNet(All)	0.113 ± 0.001
GIN	BasisNet(All)	0.110 ± 0.001
GIN	SPE(All)	0.108 ± 0.001
$\mathcal{A}_2^{\mathsf{Ngbh+}}$ (ours)	SPE(All)	0.094 ± 0.001
$\mathcal{A}_2^{\mathsf{LN+VT}}$ (ours)	SPE(All)	0.090 ± 0.001
$\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ (ours)	SPE(All)	0.087 ± 0.001

of model size and time complexity, our models are comparable to graphGPS Rampasek et al. (2022) (SOTA graph transformer): our models have similar numbers of parameters as GPS. In terms of running time: local neighbor, virtual tuple attention $\mathcal{A}_2^{\mathsf{LN+VT}}$ and simplicial transformers have similar running time as GPS ($\sim$ 20s/epoch). Our slowest variant $\mathcal{A}_2^{\mathsf{Ngbh}}$ is only about two times slower than GPS. See Appendix E for full details.

6 CONCLUDING REMARKS

In this work, we theoretically analyze the expressive power of higher-order graph transformers and systematically explore the design space for efficient and expressive high-order graph transformers. We propose sparse high-order attention mechanisms that enjoy both low computational complexity and high expressivity. Moreover, the theoretical results and architectural designs can be naturally extended to simplicial transformers. We provide preliminary experimental results to verify the performance of our high-order graph transformers and their scalability.

For future work, we note that most existing theoretical analysis of graph neural networks and their higher-order analysis center around expressiveness w.r.t. k-WL hierarchies. However, expressive power is only one interesting factor and may not be able to capture other dimensions of the effectiveness of a graph model. For example, while a graph transformer does not have more expressive power than a standard MPNN, it is believed to be more effective in capturing certain long-range interactions. It will be interesting to explore other ways to measure the power of graph learning models and study the pros and cons of transformer vs. non-transformer models.

Acknowledgements

This work was supported in part by the U.S. Army Research Office under Army-ECASE award W911NF-07-R-0003-03, the U.S. Department Of Energy, Office of Science, IARPA HAYSTAC Program, CDC-RFA-FT-23-0069, NSF Grants #2205093, #2146343,#2134274, #2112665 and #2310411.

References

Alberti, S., Dern, N., Thesing, L., and Kutyniok, G. (2023). Sumformer: Universal approximation for efficient transformers. *ArXiv*, abs/2307.02301.

Azizian, W. and Lelarge, M. (2020). Expressive power of invariant and equivariant graph neural networks. In *International Conference on Learning Representations*.

Beaini, D., Passaro, S., L'etourneau, V., Hamilton, W. L., Corso, G., and Lio', P. (2020). Directional graph networks. In *International Conference on Machine Learning*.

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *ArXiv*, abs/2004.05150.

Bo, D., Shi, C., Wang, L., and Liao, R. (2023). Specformer: Spectral graph neural networks meet transformers. *ArXiv*, abs/2303.01028.

Bodnar, C., Frasca, F., Otter, N., Wang, Y. G., Lio', P., Montúfar, G., and Bronstein, M. M. (2021a). Weisfeiler and lehman go cellular: Cw networks. In *Neural Information Processing Systems*.

Bodnar, C., Frasca, F., Wang, Y. G., Otter, N., Montúfar, G., Lio', P., and Bronstein, M. M. (2021b).

Weisfeiler and lehman go topological: Message passing simplicial networks. *ArXiv*, abs/2103.03212.

Bouritsas, G., Frasca, F., Zafeiriou, S., and Bronstein, M. M. (2023). Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668.

Bresson, X. and Laurent, T. (2017). Residual gated graph convnets. *ArXiv*, abs/1711.07553.

Briand, E. (2004). When is the algebra of multisymmetric polynomials generated by the elementary multisymmetric polynomials.

Cai, C., Hy, T. S., Yu, R., and Wang, Y. (2023). On the connection between mpnn and graph transformer. *ArXiv*, abs/2301.11956.

Cai, J.-Y., Fürer, M., and Immerman, N. (1989). An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410.

Chen, G., Chen, P., Hsieh, C.-Y., Lee, C.-K., Liao, B., Liao, R., Liu, W., Qiu, J., Sun, Q., Tang, J., et al. (2019). Alchemy: A quantum chemistry dataset for benchmarking ai models. *arXiv preprint arXiv:1906.09427*.

Chen, Z., Chen, L., Villar, S., and Bruna, J. (2020). Can graph neural networks count substructures? *CoRR*, abs/2002.04025.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L. J., and Weller, A. (2020). Rethinking attention with performers. *ArXiv*, abs/2009.14794.

Correia, G. M., Niculae, V., and Martins, A. F. T. (2019). Adaptively sparse transformers. In *Conference on Empirical Methods in Natural Language Processing*.

Corso, G., Cavalleri, L., Beaini, D., Lio', P., and Velickovic, P. (2020). Principal neighbourhood aggregation for graph nets. *ArXiv*, abs/2004.05718.

Cybenko, G. V. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

Dey, T. K. and Wang, Y. (2022). *Computational Topology for Data Analysis*. Cambridge University Press. 452 pages.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *ArXiv*, abs/2003.00982.

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. (2021). Graph neural networks with learnable structural and positional representations. *ArXiv*, abs/2110.07875.

Dwivedi, V. P., Rampáek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. (2022). Long range graph benchmark. *ArXiv*, abs/2206.08164.

Ebli, S., Defferrard, M., and Spreemann, G. (2020). Simplicial neural networks. *ArXiv*, abs/2010.03633.

Geerts, F. (2020a). The expressive power of kth-order invariant graph networks. *ArXiv*, abs/2007.12035.

Geerts, F. (2020b). Walk message passing neural networks and second-order graph neural networks. *ArXiv*, abs/2006.09499.

Ghani, R. (2001). Cmu. world wide knowledge base (web-kb) project.

Giusti, L., Battiloro, C., Lorenzo, P. D., Sardellitti, S., and Barbarossa, S. (2022). Simplicial attention neural networks. *ArXiv*, abs/2203.07485.

Goh, C. W. J., Bodnar, C., and Lio', P. (2022). Simplicial attention networks. *ArXiv*, abs/2204.09455.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *ArXiv*, abs/2005.00687.

Huang, Y., Lu, W., Robinson, J., Yang, Y., Zhang, M., Jegelka, S., and Li, P. (2023). On the stability of expressive positional encodings for graph neural networks. *arXiv preprint arXiv:2310.02579*.

Hussain, M. S., Zaki, M. J., and Subramanian, D. (2021). Global self-attention as a replacement for graph convolution. *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*.

Kim, J., Nguyen, T. D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. (2022). Pure transformers are powerful graph learners. *ArXiv*, abs/2207.02505.

Kim, J., Oh, S., and Hong, S. (2021). Transformers generalize deepsets and can be extended to graphs and hypergraphs. *ArXiv*, abs/2110.14416.

Kipf, T. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907.

Kitaev, N., Kaiser, L., and Levskaya, A. (2020). Reformer: The efficient transformer. *ArXiv*, abs/2001.04451.

Knill, O. (2023). Spectral monotonicity of the hodge laplacian. *ArXiv*, abs/2304.00901.

Kreuzer, D., Beaini, D., Hamilton, W. L., L'etourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. *ArXiv*, abs/2106.03893.

Lim, D., Robinson, J., Zhao, L., Smidt, T., Sra, S., Maron, H., and Jegelka, S. (2022). Sign and basis invariant networks for spectral graph representation learning. *arXiv preprint arXiv:2202.13013*.

Ma, J., Wang, Y., and Wang, Y. (2023a). Laplacian canonization: A minimalist approach to sign and basis invariant spectral embedding. *arXiv preprint arXiv:2310.18716*.

Ma, L., Lin, C., Lim, D., Romero-Soriano, A., Dokania, P. K., Coates, M., Torr, P. H. S., and Lim, S. N. (2023b). Graph inductive biases in transformers without message passing. *ArXiv*, abs/2305.17589.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. (2019). Provably powerful graph networks. *ArXiv*, abs/1905.11136.

Maron, H., Ben-Hamu, H., Shamir, N., and Lipman, Y. (2018). Invariant and equivariant graph networks. *ArXiv*, abs/1812.09902.

Morris, C., Rattan, G., and Mutzel, P. (2020). Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. In *Advances in Neural Information Processing Systems*.

Muller, L., Galkin, M., Morris, C., and Rampávsek, L. (2023). Attending to graph transformers. *ArXiv*, abs/2302.04181.

Rampasek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. *ArXiv*, abs/2205.12454.

Roddenberry, T. M. and Segarra, S. (2019). Hodgenet: Graph neural networks for edge data. *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 220–224.

Roy, A., Saffar, M. T., Vaswani, A., and Grangier, D. (2020). Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68.

Sanford, C., Hsu, D., and Telgarsky, M. (2023). Representational strengths and limitations of transformers. *ArXiv*, abs/2306.02896.

Segol, N. and Lipman, Y. (2019). On universal equivariant set networks. *ArXiv*, abs/1910.02421.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., and Sinop, A. K. (2023). Exphormer: Sparse transformers for graphs. *ArXiv*, abs/2303.06147.

Spielman, D. (2009). Spectral graph theory. In *lecture notes*.

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NIPS*.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio', P., and Bengio, Y. (2017). Graph attention networks. *ArXiv*, abs/1710.10903.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *ArXiv*, abs/1810.00826.

Yang, M. and Isufi, E. (2023). Convolutional learning on simplicial complexes. *ArXiv*, abs/2301.11163.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. (2021). Do transformers really perform bad for graph representation? In *Neural Information Processing Systems*.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. (2020). Big bird: Transformers for longer sequences. *ArXiv*, abs/2007.14062.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. (2018). Deep sets.

Zhao, L., Jin, W., Akoglu, L., and Shah, N. (2021). From stars to subgraphs: Uplifting any gnn with local structure awareness. *ArXiv*, abs/2110.03753.

Zhou, C., Wang, X., and Zhang, M. (2023a). Facilitating graph neural networks with random walk on simplicial complexes. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhou, C., Wang, X., and Zhang, M. (2023b). From relational pooling to subgraph gnns: A universal framework for more expressive graph neural networks. In *International Conference on Machine Learning*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [**Yes**/No/Not Applicable]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [**Yes**/No/Not Applicable]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes/No/Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [**Yes**/No/Not Applicable]
   (b) Complete proofs of all theoretical results. [**Yes**/No/Not Applicable]
   (c) Clear explanations of any assumptions. [**Yes**/No/Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [**Yes**/No/Not Applicable]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [**Yes**/No/Not Applicable]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [**Yes**/No/Not Applicable]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [**Yes**/No/Not Applicable]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [**Yes**/No/Not Applicable]
   (b) The license information of the assets, if applicable. [**Yes**/No/Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [**Yes**/No/Not Applicable]
   (d) Information about consent from data providers/curators. [**Yes**/No/Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Yes/No/**Not Applicable**]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Yes/No/**Not Applicable**]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Yes/No/**Not Applicable**]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Yes/No/**Not Applicable**]

A RELATED WORK

Sparse Transformers. Transformers have achieved great success in natural language processing and, more recently, computer vision tasks. Since Transformers were proposed, researchers have been making an effort to reduce the computational complexity through linear or sparse attention mechanisms. It is noticeable that most relevant works focus on transformers for NLP and are not specially designed for graph learning. These methods can be broadly classified into two categories: (1) computing internal sparse attention through architecture design, which reduces the complexity; (2) computing full attention while encouraging the models to learn sparse patterns. The former categories try to capture global information at a lower computation cost. In comparison, the latter categories do not reduce complexity; instead, they expect the learned patterns are sparse so that the extracted representations contain important features of the data, which also eases interpretability.

The first class of internal sparse transformers have low complexity via designing their algorithms and mechanisms to compute attention scores. In the following complexity analysis, the length of sequence is denoted as n, while we ignore the hidden dimension d. Linformer (Wang et al., 2020) proposes to project n keys and values into k groups, so that the complexity is reduced from $O(n^2)$ to $O(n \times k)$. In practice, however, $\frac{n}{k}$ is usually set to a constant factor like 4, so Linformer actually reduces the complexity by a constant scale, while revealing worse performance than full attention. Routing Transformer (Roy et al., 2020) only computes attention between queries and a small subset of their nearest keys found by k-means, reducing the complexity to $O(n^{1.5})$. Similarly, Reformer (Kitaev et al., 2020) also computes local attention, except that the nearest keys are sorted by locality sensitive hashing (LSH), which results in a complexity of $O(n \log n)$. Longformer (Beltagy et al., 2020) introduces a localized sliding window based mask and a few global mask to reduce computation. Moreover, BigBird (Zaheer et al., 2020) combines three types of sparse attention mechanisms: random attention, sliding window attention, and attention with global tokens. Further, Performer (Choromanski et al., 2020) uses the Fast Attention through Positive Orthogonal Random features (FAVOR+) mechanism to approximate Softmax kernels in attention, which reduces the computation to linear complexity. Linear transformer (Katharopoulos et al., 2020) also applies linearized attention with kernel tricks, which approximates another nonlinear activation kernel.

Now we turn to the second category of transformers, which compute full attention but encourage the models to discover sparse patterns. Since reducing complexity is our main purpose, we will not pay much attention on this type. As a representative model, Correia et al. (2019) enables attention heads to have flexible, context-dependent sparse patterns, which is achieved by replacing softmax with α-entmax: a differentiable generalization of softmax that allows low-scoring items to be assigned precisely zero weight.

Graph Transformers. Graph transformers have recently achieved great attention. Since global self-attention across nodes is unable to reflect graph structures, there are a number of works exploring graph-specific architectural designs. For example, GAT (Velickovic et al., 2017) restricts attention within local neighboring nodes, and Graphormer (Ying et al., 2021) injects edge information into the attention mechanism via attention bias. More recently, some graph transformers (Rampasek et al., 2022; Bo et al., 2023; Ma et al., 2023b; Shirzad et al., 2023) have achieved even greater success with State-of-the-Art performance in a variety of tasks. GPS (Rampasek et al., 2022) combines the attention mechanism with message passing and positional/structure encodings, while Specformer (Bo et al., 2023) embeds spectral features into graph transformers. GRIT (Ma et al., 2023b) builds a transformer architecture without message passing, which consists of learned relative positional encodings initialized with random walk probabilities and a flexible attention mechanism that updates node and node-pair representations. However, while there are a great number of high-order graph networks like k-IGN (Maron et al., 2018), high-order graph transformers have been rarely studied besides primary results in Kim et al. (2021, 2022). People also have limited understanding towards the theoretical expressive power of graph transformers, especially for high-order cases.

Simplicial Networks. Besides applying transformers directly on higher order k-tuples, there is another direction worth exploring, namely higher order simplicial complexes. Instead of all k-tuples, simplicial networks and simplicial transformers focus on k-order (directed) simplices that are usually much more sparse. For example, there are always n^2 2-tuples in a graph with n vertex, while there are m 2-simplices in the simplicial complex extended from the vertex of the original (directed) graph, where $m \leq n^2$ is the number of edges in the graph. The internal sparsity nature of simplices indicates the advantages of attention computed within higher-order simplices over higher-order tuples.

Early simplicial networks are mainly based on message-passing or convolution mechanisms. Roddenberry and Segarra (2019) is the first to generalize GNN to 1-simplicial (edge) data. Bodnar et al. (2021b); Yang and Isufi (2023) formulate the message-passing and convolutional networks on simplices and simplicial complexes. Bodnar et al. (2021a) further generalize the message-passing and convolutional networks to cellular complexes. In comparison, attention-based methods on simplicial complexes are less complete. Goh et al. (2022); Giusti et al. (2022) still restrict their attention within the scope of upper and lower adjacent simplices. There is no current work that computes full attention across all k-simplices (and more generally simplices of different orders), nor has the expressive power of these simplicial attention networks been analyzed.

High Order Transformers. Kim et al. (2022) proposed a possible form of high-order transformers, which computes attention for input order-k tensors $\boldsymbol{X} \in \mathbb{R}^{n^k}$ as follows,

$$\mathrm{Attn}(\boldsymbol{X})_{\boldsymbol{j}} = \sum_{h=1}^{H} \sum_{\boldsymbol{i}} \alpha^h_{\boldsymbol{i},\boldsymbol{j}} \boldsymbol{X}_{\boldsymbol{i}} V^h O^h \tag{14}$$

$$\alpha^h = \mathrm{softmax}\left(\frac{\boldsymbol{X} Q^h (\boldsymbol{X} K^h)^\top}{\sqrt{d_k}} \right) \tag{15}$$

where the attention $\alpha^h \in \mathbb{R}^{n^k \times n^k}$, and $Q^h, K^h \in \mathbb{R}^{d \times d_k}, V^h \in \mathbb{R}^{d \times d_v}, O^h \in \mathbb{R}^{d_v \times d}$ are learnable weights.

Kim et al. (2022) proves that with augmented node and type identifiers as input, the above attention can approximate any equivalence class basis tensor $\mathbf{B}^\mu \in R^{n^{2k}}$ of linear equivariant layer $L_{k \to k}$ (Maron et al., 2018) arbitrary well. Consequently, Kim et al. (2022) concludes that a k-Transformer with node and type identifiers is at least as expressive as k-IGN, and hence k-WL. In our work, we incorporate this type of architecture into a larger family of transformers. We also show that a slightly modified transformer can simulate k-WL in a different way from approximating the equivalence class basis as described in Kim et al. (2022).

In the previously described transformer, the input is a high-order tensor $\boldsymbol{X} \in \mathbb{R}^{n^k}$, while the parameters Q^h, K^h, V^h, O^h are actually the same as the standard transformer. Sanford et al. (2023) propose another form of "high-order" transformer, which has the same $\boldsymbol{X} \in \mathbb{R}^n$ input tokens as in the standard transformer, but is parameterized with high-order weight.

Recall the notations for the *column-wise Kronecker product*. For vectors $\boldsymbol{v}^1 \in \mathbb{R}^{n_1}, \boldsymbol{v}^2 \in \mathbb{R}^{n_2}$, their *Kronecker product* $\boldsymbol{v}^1 \otimes \boldsymbol{v}^2 \in \mathbb{R}^{n_1 n_2}$ is defined as $(\boldsymbol{v}^1 \otimes \boldsymbol{v}^2)_{i_1 - 1} n_2 + i_2 = \boldsymbol{v}^1_{i_1} \boldsymbol{v}^2_{i_2}$. The *column-wise Kronecker product* of matrices $\boldsymbol{A}^1 \in \mathbb{R}^{n_1 \times m}$ and $\boldsymbol{A}^2 \in \mathbb{R}^{n_2 \times m}$ is defined as

$$\boldsymbol{A}^1 \star \boldsymbol{A}^2 = \left[\boldsymbol{A}^1_1 | \dots | \boldsymbol{A}^1_m \right] \star \left[\boldsymbol{A}^2_1 | \dots | \boldsymbol{A}^2_m \right] = \left[\boldsymbol{A}^1_1 \otimes \boldsymbol{A}^2_1 | \dots | \boldsymbol{A}^1_m \otimes \boldsymbol{A}^2_m \right] \in \mathbb{R}^{n_1 n_2 \times m} \tag{16}$$

The s-order self-attention in Sanford et al. (2023) is then defined as follows (Definition 7 in Sanford et al. (2023)). For order $s \geq 2$, input dimension d, output dimension d', and weight matrices $Q, K^1, \dots, K^{s-1} \in \mathbb{R}^{d \times d_k}$, $V^1, \dots, V^{s-1} \in \mathbb{R}^{d \times d'}$, an *$s$-order self-attention unit* is a function $f_{Q,K,V} : \mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d'}$ defined as

$$f_{Q,K,V}(\boldsymbol{X}) = \mathrm{softmax}\Big(\underbrace{\boldsymbol{X} Q}_{\in \mathbb{R}^{n \times d_k}} \underbrace{\left((\boldsymbol{X} K^1) \star \dots \star (\boldsymbol{X} K^{s-1}) \right)^\top}_{\in \mathbb{R}^{d_k \times n^{s-1}}} \Big) \underbrace{\left((\boldsymbol{X} V^1) \star \dots \star (\boldsymbol{X} V^{s-1}) \right)}_{\in \mathbb{R}^{n^{s-1} \times d'}} \tag{17}$$

Sanford et al. (2023) gives some primary analysis on the representation strength of the above high-order transformer over standard transformer (referred as 2-order in their paper, yet 1-order in our framework). Since the transformers in (Sanford et al., 2023) are different from our formulation, we leave relevant discussions in Appendix D.

B PROOF AND ADDITIONAL RESULTS

B.1 Proof and Additional Results for Section 3

B.1.1 Notations and Preliminaries

To start with, we first recall the definitions of k-neighbors in k-WL and k-FWL. For k-WL, recall

$$\mathcal{N}_j(\boldsymbol{i}) = \Big\{(i_1, \ldots, i_{j-1}, i', i_{j+1}, \ldots, i_k)\Big| i' \in [n]\Big\}. \tag{18}$$

$\mathcal{N}_j(\boldsymbol{i}), j \in [k]$ is the j-th neighbor of tuple $\boldsymbol{i}$ in WL algorithm, which is a set of n different k-tuples. Each tuple $\boldsymbol{i}$ has k such k-neighbors, and during the update stage of the k-WL algorithm, these k neighbors are aggregated as an **ordered set** of multisets:

$$\text{WL}: \boldsymbol{C}_{\boldsymbol{i}}^{t+1} = \text{hash}\Big(\boldsymbol{C}_{\boldsymbol{i}}^t, \Big(\{\{\boldsymbol{C}_{\boldsymbol{v}}^t \big| \boldsymbol{v} \in \mathcal{N}_j(\boldsymbol{i})\}\}\Big| j \in [k]\Big)\Big) \tag{19}$$

Note that this expression is equivalent to Equation (1) in the main text. There is another form of graph isomorphic algorithm called Folklore Weisfeiler-Lehman (FWL) test, and the difference between k-WL and k-FWL lies in their tuple color update process. Concretely, the neighbors for k-FWL (Maron et al., 2019) are defined as

$$\mathcal{N}_j^F(\boldsymbol{i}) = \Big((j, i_2, \ldots, i_k), (i_1, j, \ldots, i_k), \ldots, (i_1, \ldots, i_{k-1}, j)\Big) \tag{20}$$

$\mathcal{N}_j^F(\boldsymbol{i}), j \in [n]$ is the j-th neighborhood of tuple $\boldsymbol{i}$ used by FWL, which is an ordered set of k different k-tuples. Each tuple has n such k-neighbors, while they are aggregated as a **multi-set** of ordered sets in FWL update rule:

$$\text{FWL}: \boldsymbol{C}_{\boldsymbol{i}}^{t+1} = \text{hash}\Big(\boldsymbol{C}_{\boldsymbol{i}}^t, \Big\{\Big\{\Big(\boldsymbol{C}_{\boldsymbol{v}}^t \big| \boldsymbol{v} \in \mathcal{N}_j^F(\boldsymbol{i})\Big)\Big| j \in [n]\Big\}\Big\}\Big) \tag{21}$$

We clarify the following technical details following (Maron et al., 2019), which are applied in most of the relevant papers on expressive power. We adopt these techniques and conclusions throughout our proof. The techniques explain how neural networks (including transformer variants) can implement WL-like algorithms.

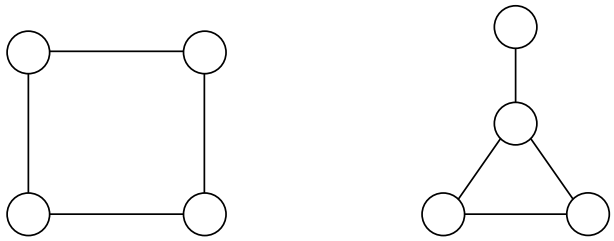
- (Remark 1) Color representation. We use tensors to represent colors. Concretely, the color of the k-tuple $\boldsymbol{i} \in [n]^k$ is represented by a vector $\boldsymbol{x} \in \mathbb{R}^d$ for some latent dimension d, and the entire color set of all k-tuples is $\boldsymbol{X} \in \mathbb{R}^{n^k \times d}$.

- (Remark 2) Multiset representation. Unlike tuples, the multiset should be invariant to the order of nodes, i.e., $g \cdot \boldsymbol{X}$ should be the same for all permutations $g \in S_n$, where S_n is the symmetric group. As pointed out by (Maron et al., 2019), *Power-sum Multi-symmetric Polynomials* (PMP) (Briand, 2004) are S_n invariant, thus can be used to encode multisets. As shown in (Maron et al., 2019), PMP generates a unique representation of each multiset, which enables us to represent multisets as tensors.

- (Remark 3) Hash function. Suppose that there are two color tensors $\boldsymbol{C} \in \mathbb{R}^{n^k \times a}, \boldsymbol{C}' \in \mathbb{R}^{n^k \times b}$, the hash function in Equation (19) and Equation (21) can be implemented through a simple concatenation, that is, the new tensor representation for each k-tuple $\boldsymbol{i}$ of color pair $(\boldsymbol{C}_i, \boldsymbol{C}'_i)$ is simply $(\boldsymbol{C}, \boldsymbol{C}') \in \mathbb{R}^{n^k \times (a+b)}$.

- (Remark 4) Input and initialization. We adopt the same input and initialization of k-tuples as in Appendix C.1 of (Maron et al., 2019), which can faithfully represent the isomorphism type of each k-tuple. That is, two tuples have identical initialization if and only if they have the same isomorphism types.

- (Remark 5) Update step of WL-like algorithm. In every update step of WL-like algorithms, when we use PMP to represent tuple colors, the color representation of a new multiset $\{\{\boldsymbol{j}\}\}$ consisting of tuples $\boldsymbol{j}$ with known colors $\boldsymbol{B}_{\boldsymbol{j}}$ can be implemented by the summation of polynomial transform over $\boldsymbol{B}_{\boldsymbol{j}}$, i.e. $\boldsymbol{C}(\{\{\boldsymbol{j}\}\}) = \sum_{\boldsymbol{j}} \tau(\boldsymbol{B}_{\boldsymbol{j}})$, where τ is a specifically designed polynomial function that is injective, see (Maron et al., 2019) for more details. In practical networks, the polynomial function can be replaced by an MLP using the universal approximation power of MLPs (Hornik, 1991). When the approximation power is sufficiently small, the injective properties can still be preserved to achieve the upper bound of the theoretical expressive power (Maron et al., 2019). Most relevant papers adopt this assumption, and we follow this as well in this paper. However, we emphasize that practical networks may not achieve their full expressive power when they



(a) G (b) H

Figure 2: A pair of non-isomorphic graphs that can be distinguished by 1-WL and 2-WL, but cannot be distinguished by $\mathcal{A}_1$ and $\mathcal{A}_2$. For $k \geq 3$, both k-WL and $\mathcal{A}_k$ can distinguish them.

are parameterized with learnable MLPs and embedding layers, other than strictly using PMP and polynomial functions. As for new tuples consisting of known tuples, as discussed above, the new color representation is simply the concatenation of known tuple colors.

- (Remark 6) Histogram computation. The final step to determine the graph isomorphism is the histogram of tuple colors $H(\boldsymbol{B})$. Suppose the number of colors is b (which is finite for finite graphs); we apply a tuple-wise MLP $m : \mathbb{R}^d \to \mathbb{R}^b$ mapping each tuple color to a one-hot vector in $\mathbb{R}^b$. Summing over the one-hot vectors via a summing-invariant operator $h : \mathbb{R}^{n^k \times b} \to \mathbb{R}^b$, we obtain an injective and permutation-invariant representation of the color histogram: $H(\boldsymbol{B}) = h(m(\boldsymbol{B}))$.

In addition, while considering the approximation power of neural networks, we always assume that the inputs and network weights are compact. Specifically, for input $\boldsymbol{X}$, we have $\forall \boldsymbol{i} \in [n]^k, ||\boldsymbol{X}_{\boldsymbol{i}}|| < C_1$, and for weight matrices of the transformer, we have $||Q|| < C_2, ||K|| < C_2, ||V|| < C_2$. This is a mild and proper condition on feature space and parameter space which generally holds for practical networks.

B.1.2 Approximating Power and Theoretical Expressive Power

Now we prove the results of the expressive power of higher-order transformers.

Theorem B.1. (Theorem 3.2 in the main text.) *Without additional input, $\mathcal{A}_k$ is strictly less expressive than k-WL and k-IGN.*

Proof. The proof is straightforward. Without any extra inputs (including positional encodings, structural encodings, and tuple indices), the initialization of every k-tuple is the same as in k-WL, see Maron et al. (2019) for more details. Then according to Definition 3.1 in the main text, tuples with identical initialization representations (colors) always obtain identical updates, suggesting that the color histogram always remains the same as the initial one. Consequently, the expressive power of $\mathcal{A}_k$ is the same as the 0-th step (initialization) of k-WL. There exist non-isomorphic graphs that can be distinguished by stable k-WL histograms but not its initialization, yet the other direction does not hold. Therefore, $\mathcal{A}_k$ is strictly less expressive than k-WL. □

As a corollary, since the initialization of $(k+1)$-WL is always strictly more powerful than the initialization of k-WL, we conclude that $\mathcal{A}_{k+1}$ is strictly more powerful than $\mathcal{A}_k$.

Corollary B.2. $\forall k \geq 1$, *$\mathcal{A}_{k+1}$ is strictly more powerful than $\mathcal{A}_k$.*

Proof. We have already shown in the above proof of Theorem 3.2 that $\mathcal{A}_k$ is as powerful as the initialization as k-WL. It is known that for all $k \geq 1$, the initialization of $k+1$-WL is strictly more powerful than the initialization of k-WL. This holds because the initialization of $(k+1)$-WL can always distinguish whether there is a $(k+1)$-clique in the graph, while the initialization of k-WL fails. □

Figure 2 provides an example of a pair of non-isomorphic graphs that can be distinguished by 1-WL and 2-WL, but cannot be distinguished by $\mathcal{A}_1$ and $\mathcal{A}_2$. Here we demonstrate the procedure. In the graph pairs, $n = 4$. In

our illustration, for simplicity, we use different alphabets $a, b, \ldots$ to represent different features $\boldsymbol{X_i}$ using a hash function.

For $k = 1$, we consider 1-WL and $\mathcal{A}_1$ (i.e., the standard first-order graph transformer). In the initialization stage, both 1-WL and $\mathcal{A}_1$ have identical 1-tuple (node) feature initialization for both graphs G and H:

$$1 - \mathrm{WL}(G)^{(0)} = \{\{a, a, a, a\}\},\ 1 - \mathrm{WL}(H)^{(0)} = \{\{a, a, a, a\}\} \tag{22}$$

$$\mathcal{A}_1(G)^{(0)} = \{\{a, a, a, a\}\},\ \mathcal{A}_1(H)^{(0)} = \{\{a, a, a, a\}\} \tag{23}$$

That is, both 1-WL and $\mathcal{A}_1$ cannot distinguish G and H through their initialization. However, after one iteration, denote $b = \mathrm{Hash}(a, \{\{a, a\}\}), c = \mathrm{Hash}(a, \{\{a\}\}), d = \mathrm{Hash}(a, \{\{a, a, a\}\})$, and e as the updated feature calculated by $\mathcal{A}_1$, we have

$$1 - \mathrm{WL}(G)^{(1)} = \{\{b, b, b, b\}\},\ 1 - \mathrm{WL}(H)^{(1)} = \{\{b, b, c, d\}\} \tag{24}$$

$$\mathcal{A}_1(G)^{(1)} = \{\{e, e, e, e\}\},\ \mathcal{A}_1(H)^{(1)} = \{\{e, e, e, e\}\} \tag{25}$$

Therefore, 1-WL can distinguish two graphs since all four nodes in G have degree 2, yet the degree histogram in H is $\{\{2, 2, 1, 3\}\}$. In comparison, $\mathcal{A}_1$ has identical representation for G and H. The histograms calculated by $\mathcal{A}_1$ are actually not updated compared with the initialization. By deduction, we can easily verify that for all iterations $t \geq 1$, we always have

$$\mathcal{A}_1(G)^{(t)} = \{\{a^{(t)}, a^{(t)}, a^{(t)}, a^{(t)}\}\},\ \mathcal{A}_1(H)^{(t)} = \{\{a^{(t)}, a^{(t)}, a^{(t)}, a^{(t)}\}\} \tag{26}$$

which implies that $\mathcal{A}_1$ always fails in distinguishing G and H. This observation is also in agreement with the fact that the plain first-order graph transformer cannot capture any edge information.

For $k = 2$ the same conclusion can be drawn. In the initialization stage, there are $n^2 = 16$ number of 2-tuples. Denote $a = (i, i)$, $b = (i, j), i \neq j, e_{ij} = e_{ji} = 1$ where $e_{ij} = e_{ji} = 1$ indicates that there is an undirected edge between the nodes i, j, and finally $c = (i, j), i \neq j, e_{ij} = e_{ji} = 0$ which implies that i, j are not connected. Thus,

$$2 - \mathrm{WL}(G)^{(0)} = \{\{a, a, a, a, b, b, b, b, b, b, b, b, c, c, c, c\}\},\ 2 - \mathrm{WL}(H)^{(0)} = \{\{a, a, a, a, b, b, b, b, b, b, b, b, c, c, c, c\}\} \tag{27}$$

$$\mathcal{A}_2(G)^{(0)} = \{\{a, a, a, a, b, b, b, b, b, b, b, b, c, c, c, c\}\},\ \mathcal{A}_2(H)^{(0)} = \{\{a, a, a, a, b, b, b, b, b, b, b, b, c, c, c, c\}\} \tag{28}$$

After one iteration, denote

$$d = \mathrm{Hash}(a, (\{\{a, b, b, c\}\}, \{\{a, b, b, c\}\})) \tag{29}$$

$$e = \mathrm{Hash}(a, (\{\{a, b, c, c\}\}, \{\{a, b, c, c\}\})) \tag{30}$$

$$f = \mathrm{Hash}(a, (\{\{a, b, b, b\}\}, \{\{a, b, b, b\}\})) \tag{31}$$

$$g = \mathrm{Hash}(b, (\{\{a, b, b, c\}\}, \{\{a, b, b, c\}\})) \tag{32}$$

$$h = \mathrm{Hash}(b, (\{\{a, b, b, b\}\}, \{\{a, b, c, c\}\})) \tag{33}$$

$$o = \mathrm{Hash}(b, (\{\{a, b, c, c\}\}, \{\{a, b, b, b\}\})) \tag{34}$$

$$p = \mathrm{Hash}(b, (\{\{a, b, b, b\}\}, \{\{a, b, b, c\}\})) \tag{35}$$

$$q = \mathrm{Hash}(b, (\{\{a, b, b, c\}\}, \{\{a, b, b, b\}\})) \tag{36}$$

$$r = \mathrm{Hash}(c, (\{\{a, b, b, c\}\}, \{\{a, b, b, c\}\})) \tag{37}$$

$$s = \mathrm{Hash}(c, (\{\{a, b, c, c\}\}, \{\{a, b, b, c\}\})) \tag{38}$$

$$u = \mathrm{Hash}(c, (\{\{a, b, b, c\}\}, \{\{a, b, c, c\}\})) \tag{39}$$

Then 2-WL gives

$$2-\mathrm{WL}(G)^{(1)} = \{\{d,d,d,d,g,g,g,g,g,g,g,g,r,r,r,r\}\} \tag{40}$$

$$2-\mathrm{WL}(H)^{(1)} = \{\{d,d,e,f,g,g,h,o,p,p,q,q,s,s,u,u\}\} \tag{41}$$

The histogram of G and H are obvious different, therefore 2-WL can distinguish them within one iteration, which is consistent with the fact that 2-WL is as powerful as 1-WL.

In comparison, although $\mathcal{A}_k$ is able to incorporate edge information in the initialization stage, it is unable to learn further connectivity of the graph structure through the fully dense attention mechanism. Indeed, for $\forall t \geq 1$, the histogram of $\mathcal{A}_2$ after t iterations always gives

$$\mathcal{A}_2(G)^{(t)} = \{\{a^{(t)},a^{(t)},a^{(t)},a^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},c^{(t)},c^{(t)},c^{(t)},c^{(t)}\}\} \tag{42}$$

$$\mathcal{A}_2(H)^{(t)} = \{\{a^{(t)},a^{(t)},a^{(t)},a^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},b^{(t)},c^{(t)},c^{(t)},c^{(t)},c^{(t)}\}\} \tag{43}$$

hence always fail to distinguish two graphs.

For $k \geq 3$, note that there is a 3-clique in H but not in G. Therefore, both 3-WL and $\mathcal{A}_k$ can distinguish them via initialization.

In summary, a plain $\mathcal{A}_k$ is always strictly less expressive than k-WL and k-IGN in terms of distinguishing non-isomorphic graphs. However, for real-world graphs where the features are represented by a continuous-valued vector, the attention mechanism provides the potential for powerful representation learning. Particularly, with enhancements such as positional/structural encodings (PE/SE) and our sparse attention mechanisms, we can introduce asymmetry and variety into tuple features, obtaining powerful representations over graphs. See Appendix B for more theoretical analysis, and Appendix E for more implementation details and practical benefits.

Now we analyze the enhancement of augmenting inputs with tuple indices. As emphasized in our main text, this enhancement has potential drawbacks including breaking permutation invariance (which is also the case for (Kim et al., 2022)), hence is only of theoretical interest.

Theorem B.3. (Theorem 3.3 in main text.) *For inputs $\boldsymbol{X} \in \mathbb{R}^{n^k \times (d+k)}$ where each element is a concatenation of a d-dimensional tuple feature and k-dimension index of the tuple, one layer of $\mathcal{A}_k$ with hidden dimension $O(k)$ and k heads augmented with input MLPs and residual connection can approximate one k-WL iteration arbitrarily well. If the softmax function is replaced by element-wise ReLU activation, $\mathcal{A}_k$ can exactly simulate k-WL.*

Proof. We breakdown the update function of k-WL Equation (19) into several parts. The core idea of our construction is to select k-neighbors in Equation (18) via the multiplication of queries and keys, so that the update only involves the values (colors) of these neighboring tuples. As there are k different neighbors for each tuple (depending on the position of indices being replaced), they are implemented by k heads of transformer, and the concatenation of representations output by independent heads naturally implements the ordered set of neighbors as k heads are allowed to be parameterized with different weights. Additionally, the color of the previous iteration and the hash function can be implemented by residual connection and feed forward layers (see Maron et al. (2019) for more details).

First, we show that the h-th head with input MLPs $\phi_j : \mathbb{R}^{d+k} \to \mathbb{R}^{2(k-1)}$ and the softmax function $\mathrm{softmax}\big(\phi(\boldsymbol{X})Q(\phi(\boldsymbol{X})K)^\top\big)$ can approximate the selection of the h-th neighbor arbitrarily well. As we only want to reserve the index information in this part, inspired by (Sanford et al., 2023), we define the tuple-wise MLP as

$$\frac{1}{c}\phi_h(x_{\boldsymbol{i}})Q^h = \phi_h(x_{\boldsymbol{i}})K^h = \phi_h\big([z_{\boldsymbol{i}},\boldsymbol{i}]\big) = \Big[\cos(\frac{2\pi i_1}{M}),\sin(\frac{2\pi i_1}{M}),\ldots,\cos(\frac{2\pi i_{h-1}}{M}),\sin(\frac{2\pi i_{h-1}}{M}), \tag{44}$$

$$\cos(\frac{2\pi i_{h+1}}{M}),\sin(\frac{2\pi i_{h+1}}{M}),\ldots,\cos(\frac{2\pi i_k}{M}),\sin(\frac{2\pi i_k}{M})\Big] \in \mathbb{R}^{2(k-1)} \tag{45}$$

where $\boldsymbol{i}$ is the k-dimensional multi-index for the tuple, $z_{\boldsymbol{i}}$ is the d-dimensional tuple feature being dropped for queries and keys, $M \geq n$ is a constant integer and $c \in \mathbb{R}$ is a positive constant. That is to say, we drop the h-th dimension of the index and reserve the remaining $k-1$ ones to find the tuples as candidates of h-th neighbor,

who have identical indices as the query except the h-th index. Then for query tuple $\boldsymbol{i}$ and potential target tuple $\boldsymbol{j}$, we have

$$(\phi_h(x_{\boldsymbol{i}})Q^h)^\top \phi_h(x_{\boldsymbol{j}})K^h = c \sum_{l\in[k],l\neq h} \cos\Big(\frac{2\pi(i_l - j_l)}{M}\Big) \tag{46}$$

Since $M \geq n$, the above value reaches its maximum $c(k-1)$ if and only if $i_l = j_l$ for all $l \in [k], l \neq h$, which exactly corresponds to the indices of n tuples in the h-th neighbor of $\boldsymbol{i}$. Then let $c \to \infty$, the output of the softmax would be

$$\Big(\text{softmax}\Big((\phi_h(\boldsymbol{X})Q^h)(\phi_h(\boldsymbol{X})K^h)^\top\Big)\Big)_{\boldsymbol{i},\boldsymbol{j}} \to \begin{cases} \frac{1}{n}, & \boldsymbol{j} \in \mathcal{N}_h(\boldsymbol{i}) \\ 0, & \boldsymbol{j} \notin \mathcal{N}_h(\boldsymbol{i}) \end{cases} \tag{47}$$

Then the output of the h-th head attention would be

$$\Big(\text{softmax}\Big((\phi_h(\boldsymbol{X})Q^h)(\phi_h(\boldsymbol{X})K^h)^\top\Big)\boldsymbol{X}V^h\Big)_{\boldsymbol{i}} \to \frac{1}{n}\sum_{\boldsymbol{j}\in\mathcal{N}_h(\boldsymbol{i})} V^h \tag{48}$$

where the values $x_{\boldsymbol{j}}V^h$ only retain information of tuple features $z_{\boldsymbol{j}}$ and drop the indices to represent the color of tuple $\boldsymbol{j}$ in the last iteration (along with some possible injective transformation). Since n is constant, the $\frac{1}{n}$ factor does not affect the injectivity. The above results show that we can successfully represent $\mathcal{N}_h(\boldsymbol{i})$, which is a multiset of n different k-tuples. According to Remark 5, the summation operation can injectively represent the color of $\mathcal{N}_h(\boldsymbol{i})$ - although another MLP τ is needed before summing the color representations, this could be easily achieved by adding another input MLP (which is always the case whenever we need to use Remark 5).

It is remarkable that the above approximation can be exact if we replace the softmax with an element-wise ReLU activation:

$$\text{ReLU}\Big((\phi_h(x_{\boldsymbol{i}})Q^h)^\top \phi_h(x_{\boldsymbol{j}})K^h - (c(k-1-\frac{1}{M^2}))\Big) = \begin{cases} \frac{c}{M^2}, & \boldsymbol{j} \in \mathcal{N}_h(\boldsymbol{i}) \\ 0, & \boldsymbol{j} \notin \mathcal{N}_h(\boldsymbol{i}) \end{cases} \tag{49}$$

The rest of the proof is more straightforward. As we have k different heads with different parameters, we repeat the above procedure to get all the k neighbors of $\boldsymbol{i}$. The set of neighbors can be represented by concatenating the output of these heads with the fixed order $h = 1, \ldots, k$. The color of $\boldsymbol{i}$ in the previous iteration $\boldsymbol{C}_{\boldsymbol{i}}^t$ is reserved by residual connection. Then, according to Remark 3, the final hash function can be exactly implemented by concatenation of $\boldsymbol{C}_{\boldsymbol{i}}^t$ and Equation (48) (Equation (49)). □

Our construction and proof are different from (Kim et al., 2022). Instead of directly simulating k-WL as we do, they aimed to approximate equivalence class basis in k-IGN. In detail, they proposed to augment tuple features with auxiliary inputs, namely 'node identifiers' and 'type identifiers' for $\mathcal{A}_k$ transformer, so that the latter can approximate equivalence class basis in k-IGN and thus k-WL. However, compared with their constructions, our version reveals at least the following advantages: (1) The input dimension of their construction is at least $O(d+kn)$ since they use orthogonal basis for each node as node identifiers, growing with n makes it unrealistic. In comparison, our construction needs only $d+k$ input dimension and $O(k)$ hidden dimension, which are constants independent of n. Furthermore, since we need to index integers from 1 to n, we need $k \log n$ bits in total; in comparison, since orthonormal vectors can consist of $0, 1$, resulting kn bits. (2) Their node identifiers serve as explicit features instead of only indexing, making their results stochastic and not permutation invariant. Actually, the role of their identifiers is the labeling method in (Zhou et al., 2023b) to break the symmetry, but they do not perform relational pooling (summing over all possible permutations of labeling orders) to maintain permutation invariance of the models. In contrast, our output is deterministic, and if we use the exact parameterization of the ReLU version as in the proof above, the output of our transformer is permutation invariant, although generally the output is not permutation invariant for arbitrary parameterization. (3) The transformer parameters are required to be infinite in their construction, while we can avoid this problem with the ReLU activation version. We refer readers to (Kim et al., 2022) for more details of their methods.

B.2 Proof and Additional Results for Section 4

B.2.1 Kernelized Attention

Literature so far has already applied linear/kernelized attention to standard graph transformers (Rampasek et al., 2022), yet rarely to high-order graph transformers. Kim et al. (2022) indeed applied Performer to their TokenGT, but their TokenGT is actually not a strict second-order transformer since it apparently does not consider all 2-tuples. Moreover, the theoretical guarantees of the kernelized transformers are still left unexplored, especially for high-order cases. We now formally state the definition of kernelized attention. Similar to the case for standard (one-dimensional) transformer, a single head k-order self-attention layer can be reformed as

$$\boldsymbol{X}_i^{(l+1)} = \sum_{j=1}^{n^k} \frac{\kappa(Q^{(l)}\boldsymbol{X}_i^{(l)}, K^{(l)}\boldsymbol{X}_j^{(l)})}{\sum_{k=1}^{n^k}\kappa(Q^{(l)}\boldsymbol{X}_i^{(l)}, K^{(l)}\boldsymbol{X}_k^{(l)})}\dot{(}V^{(l)}\boldsymbol{X}_j^{(l)}) \tag{50}$$

where $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the softmax kernel $\kappa(\mathbf{x}, \mathbf{y}) = \exp(\mathbf{x}^T\mathbf{y})$. The kernel trick approximates the softmax via

$$\kappa(\mathbf{x}, \mathbf{y}) =< \Phi(\mathbf{x}), \Phi(\mathbf{y}) >\approx \phi(\mathbf{x})^T\phi(\mathbf{y}) \tag{51}$$

where the first equation is by Mercer's theorem and the latter is a low-dimensional approximation with random transformation $\phi : \mathbb{R}^d \to \mathbb{R}^m$. In Performer Choromanski et al. (2020), $\phi(\mathbf{x}) = \frac{\exp\left(-\frac{||X||_2^2}{2}\right)}{\sqrt{m}}[\exp(\mathbf{w}_1^T\mathbf{x}), \ldots, \exp(\mathbf{w}_m^T\mathbf{x})]$ where $\mathbf{w}_k \sim \mathcal{N}(0, I_d)$. In Linear Transformer Katharopoulos et al. (2020), $\phi(\mathbf{x}) = elu(\mathbf{x}) + 1$. Then we can further rewrite the attention as

$$\boldsymbol{X}_i = \frac{\left(\phi(\boldsymbol{X}_iQ)^\top \sum_{j=1}^{n^k}\left(\phi(\boldsymbol{X}_jK) \otimes (\boldsymbol{X}_jV)\right)\right)^\top}{\phi(\boldsymbol{X}_iQ)^\top \sum_{l=1}^{n^k}\phi(\boldsymbol{X}_lK)} \tag{52}$$

where $\otimes$ is the outer product. Note that the two summations $\sum_{j=1}^{n^k}\left(\phi(\boldsymbol{X}_jK)\otimes(\boldsymbol{X}_jV)\right) \in \mathbb{R}^{md'}$ and $\sum_{l=1}^{n^k}\phi(\boldsymbol{X}_lK)$ are shared for all query tokens, while the former has the bottleneck complexity $O(n^kmd')$. Again, since m, d' are of the same magnitude as d, the total time complexity of kernelized $\mathcal{A}_k$ is $O(n^kd^2)$.

Now we prove the approximation results of k-IGN to kernelized $\mathcal{A}_k$ in the main text.

Theorem B.4. (Theorem 4.1 in main text.) *k-IGN can approximate kernelized $\mathcal{A}_k$ with Linear Transformer or Performer architectures arbitrarily well. $\mathcal{A}_k$ with Linear Transformer or Performer architectures is strictly less powerful than k-IGN.*

Proof. First, we emphasize the fact that although having n^k input tokens, a pure $\mathcal{A}_k$ treats all tuples homogeneously regardless of equivalence classes. Consequently, the $\mathcal{A}_k$ with Linear Transformer or Performer architectures is of the same form as DeepSets and Sumformer(Alberti et al., 2023).

To show that k-IGN can approximate kernelized $\mathcal{A}_k$, we construct identical weights for all equivalence classes in k-IGN, namely $w_\mu = w, b_\lambda = b, \forall\mu \in [\text{bell}(2k)], \forall\lambda \in [\text{bell}(k)]$. Then the update function in Equation (4) (in main text) reduces to

$$L_{k\to k}(\boldsymbol{X})_{\boldsymbol{i}} = \sum_{\boldsymbol{j}} \boldsymbol{X}_{\boldsymbol{j}}w + b \tag{53}$$

Compared with MPNN of the first order, k-IGN aggregates global information from all n^k tuples by design, hence can recover the role of "virtual node" in MPNN (Cai et al., 2023). We will show that Equation (53) with residual connection and output MLPs can approximate kernelized $\mathcal{A}_k$ with Linear Transformer and Performer architectures arbitrarily well in $O(1)$ depth and $O(1)$ width. A similar technique is used in our proof of Theorem 4.4 and (Cai et al., 2023).

We now consider another k-IGN layer along with an input MLP. The input MLP computes $\phi(\boldsymbol{X_j}K) \otimes (\boldsymbol{X_j}V)$ for all $\boldsymbol{j}$, and the k-IGN layer computes $\sum_{\boldsymbol{j}} \phi(\boldsymbol{X_j}K) \otimes (\boldsymbol{X_j}V)$.

Under the compactness assumption of inputs and weight matrices (stated in Appendix B.1.1), we consider the following construction of two k-IGN layers along with residual connection and output MLPs ψ. Mathematically,

$$\boldsymbol{X}_{\boldsymbol{i}}^{(new)} := \psi\big(\boldsymbol{X_i}, \sum_{\boldsymbol{j}} \boldsymbol{X_j} w + b, \sum_{\boldsymbol{j}} \phi(\boldsymbol{X_j}K) \otimes (\boldsymbol{X_j}V)\big) = \frac{\Big(\phi(\boldsymbol{X_i}Q)^\top \sum_{\boldsymbol{j}} \big(\phi(\boldsymbol{X_j}K) \otimes (\boldsymbol{X_j}V)\big)\Big)^\top}{\phi(\boldsymbol{X_i}Q)^\top \sum_{\boldsymbol{l}} \phi(\boldsymbol{X_l}K)} \tag{54}$$

where the first equality is our construction and the second equality is the target that we aim to approximate arbitrarily well with $O(1)$ width and $O(1)$ depth. Analogously to (Cai et al., 2023), by the uniform continuity of the functions, it suffices to show that 1) we can approximate ϕ, 2) we can approximate multiplication and vector-scalar division, 3) the denominator $\phi(\boldsymbol{X_i}Q)^\top \sum_{\boldsymbol{l}} \phi(\boldsymbol{X_l}K)$ is uniformly lower bounded by a positive number for any node features.

For 1), each component of ϕ (in both Performer and Linear Transformer)is continuous, and all inputs $\boldsymbol{X_j}Q, \boldsymbol{X_j}K$ lie in compact domain. Therefore, ϕ can be approximated arbitrarily well by MLP with $O(1)$ width and $O(1)$ depth (Cybenko, 1989).

For 2), since multiplication and vector-scalar division operations are all continuous, it suffices to show that all operands lie in a compact domain. This is true since $\boldsymbol{X}^{(0)}$ and Q, K, V are all compact, ϕ is continuous and that n is fixed. Lastly, since all these operations do not involve n, the depth and width of MLPs are constant in n.

For 3), we need to show that the denominator is bound by a positive constant. In Performer, $\phi(\boldsymbol{X}) = \frac{\exp\left(-\frac{||x||_2^2}{2}\right)}{\sqrt{m}}[\exp(\mathbf{w}_1^T\boldsymbol{X}), \ldots, \exp(\mathbf{w}_m^T\boldsymbol{X})]$ where $\mathbf{w}_k \sim \mathcal{N}(0, I_d)$. As $||\mathbf{w}_i^T\boldsymbol{X}|| \leq ||\mathbf{w}_i|| \cdot ||\boldsymbol{X}||$, which implies that $\exp(\mathbf{w}_i^T\boldsymbol{X})$ is lower bounded by $\exp(-||\mathbf{w}_i|| \cdot ||\boldsymbol{X}||)$. Consequently, the demonimator $\phi(\boldsymbol{X_i}Q)^\top \sum_{\boldsymbol{l}} \phi(\boldsymbol{X_l}K)$ is lower bounded. For Linear Transformer, the proof is essentially the same as Performer. It boils down to showing that $\phi(\mathbf{x}) = elu(\mathbf{x}) + 1$ is continuous and positive, which is indeed the case.

In conclusion, k-IGN along with residual connection and output MLPs can approximate kernelized $\mathcal{A}_k$ with Linear Transformer or Performer architectures arbitrarily well.

For the second part that kernelized $\mathcal{A}_k$ is strictly less expressive than k-IGN, recall that we have already shown that even $\mathcal{A}_k$ without kernelization is strictly less expressive than k-WL (and thus k-IGN). Kernelized $\mathcal{A}_k$ (either with Linear Transformer or Performer architectures) cannot be more expressive than $\mathcal{A}_k$ without kernelization as we can always find a set of parameters of $\mathcal{A}_k$ to implement a given kernelized $\mathcal{A}_k$. Consequently, kernelized $\mathcal{A}_k$ with Linear Transformer or Performer architectures is strictly less powerful than k-IGN. □

Note that Linear equivalence layers in Equation (53) augmented with input and output MLPs suffices a Sumformer (Alberti et al., 2023) by definition, which can simulate $\mathcal{A}_k$ (without kernelization) within $O(1)$ depth and $\binom{n^k+d}{d} - 1$ width for continuous function (Alberti et al., 2023). The same conclusion can be drawn using the concept of Deepset (Zaheer et al., 2018; Segol and Lipman, 2019). Although the computation complexity is not practical (note that k-IGN and k-PPGN require the same complexity to achieve full expressivity as well (Maron et al., 2019)), the theoretical implication is interesting: it verifies our previous conclusion that $\mathcal{A}_k$ cannot be more expressive than k-IGN.

B.2.2 Sparse Attention Mechanisms

As we emphasize in the main texts, the plain global attention $\mathcal{A}_k$ has limited expressive power due to the homogeneous dense attention scheme (Theorem 3.2). We now provide proofs for the expressive power of our proposed sparse attention mechanisms.

Neighbor Attention. Analogous to k-WL, neighbor attention mechanism benefits from locality and sparsity, revealing expressive power with a lower bound of k-WL. We first restate the definition of neighbor atention,

$$\Big(\mathcal{A}_k^{\mathsf{Ngbh}}(\boldsymbol{X}, \boldsymbol{X})\Big)_{\boldsymbol{i}} = \text{Concat}\Big[\text{softmax}\Big((x_{\boldsymbol{i}}Q^j)\big(x_{[\psi_j(\boldsymbol{i},u)|u\in[n]]}K^j\big)^\top\Big)\big(x_{[\psi_j(\boldsymbol{i},u)|u\in[n]]}V^j\big)\Big|j \in [k]\Big] \tag{55}$$

where $\psi_j(\boldsymbol{i}, u)$ means replacing the j-th element in $\boldsymbol{i}$ with u, and thus $x_{[\psi_j(\boldsymbol{i},u)|u\in[n]]} \in \mathbb{R}^{n\times d},\ j \in [k]$ is just the feature of the j-th neighbor of tuple $\boldsymbol{i}$.

Theorem B.5. (Theorem 4.2 in main text.) *Neighbor attention $\mathcal{A}_k^{\mathsf{Ngbh}}$ with residual connection, output MLPs and k heads is as powerful as k-WL.*

Proof. We first show that $\mathcal{A}_k^{\mathsf{Ngbh}}$ simulates k-WL by constructing a set of parameter weights. Define

$$x_{\boldsymbol{i}}Q^j = \vec{1}, \boldsymbol{i} \in \mathbb{R}^{n^k}, j \in [k] \tag{56}$$

$$x_{\boldsymbol{i}}K^j = \vec{1}, \boldsymbol{i} \in \mathbb{R}^{n^k}, j \in [k] \tag{57}$$

where $Q^j, K^j, V^j (j \in [k])$ are weight parameters of the j-th head. Note again that we always allow bias terms for Q, K, V matrices in practical transformers; thus, the above equation can be easily obtained by setting weight matrices as zero matrices and setting the bias as $\vec{1} \in \mathbb{R}^{d_k}$, where d_k is the hidden dimension of query and keys which is a constant.

Then the neighbor attention reduces to

$$\left(\mathcal{A}_k^{\mathsf{Ngbh}}(\boldsymbol{X}, \boldsymbol{X})\right)_{\boldsymbol{i}} = \text{Concat}\left[\frac{1}{n}\sum_{u\in[n]} x_{\psi_j(\boldsymbol{i},u)}V^j \Big| j \in [k]\right] \tag{58}$$

In graph-level isomorphism task, we consider a fixed graph pair, thus n is a fixed constant for our interested graphs, which will not affect the ability to implement hash function. Therefore, the neighbor attention aggregates information from tuples in j-th k-neighbor through the j-th head, for $j \in [k]$ respectively, which is exactly what k-WL does in Equation (19). Then according to (Maron et al., 2019) (and similar to the techniques we used in Section 3), the concat operation, residual connection and output MLPs implement an ordered set (), $c_k^{t-1}(\boldsymbol{v}, G)$ and the hash function in Equation (1) (in the main text) or Equation (19), thus completely simulating k-WL with proper constructions.

The other side is because the attention described in Equation (8) (in the main text) is always an instance of Equation (1) in the main text as long as we restrict the reception field to the k-neighbors.

□

In other words, $\mathcal{A}_k^{\mathsf{Ngbh}}$ is actually a form of k-WL implementation. Although we need more layers of $\mathcal{A}_k^{\mathsf{Ngbh}}$ (and more k-WL iterations) to simulate one layer k-IGN, the neighbor attention is much lower in complexity while having the same expressive power. Particularly, since one $\mathcal{A}_k^{\mathsf{Ngbh}}$ layer is equivalent to one k-WL iteration, and that $(k-1)$ iterations of k-WL simulates one k-IGN layer (Geerts, 2020a). Therefore, we need $(k-1)$ layers of $\mathcal{A}_k^{\mathsf{Ngbh}}$ to simulate one k-IGN layer, leading to $O(n^{k+1}k(k-1)d)$ complexity (recall that in $\mathcal{A}_k^{\mathsf{Ngbh}}$ each k-tuple computes attention with k k-neighbors); it is still more efficient than one k-IGN with $O(n^{2k}d)$ complexity, though. Additionally, as we discuss in the main text, although attention does not increase expressive power compared to k-WL, it may improve real-world task performance due to more flexible attention-based aggregation compared to naive homogeneous aggregation. A natural extension would be a dense $\mathcal{A}_k$ with complete tuple adjacent information or equivalence class basis is as powerful as k-WL and thus k-IGN, yet the attention may lead to stronger real-world performance.

Local Neighbor Attention. Now we move to the local neighbor attention $\mathcal{A}_k^{\mathsf{LN}}$. Before we state our attention mechanism, we first give a brief summary of the algorithms in (Morris et al., 2020).

Morris et al. (2020) proposed a family of δ-k-dimensional WL algorithms, which is more powerful than k-WL. Additional to the k-neighbor in k-WL, δ-k-WL augments each j-th k-neighbor with the connectivity between the node being replaced $\boldsymbol{i}_j$ and the n nodes replacing it. Formally,

$$\delta\text{-}k\text{-WL}: \boldsymbol{C}_{\boldsymbol{i}}^{t+1} = \text{hash}\left(\boldsymbol{C}_{\boldsymbol{i}}^t, \left(\{\{\boldsymbol{C}_{\boldsymbol{j}}^t, \text{adj}(\boldsymbol{i}_j, \boldsymbol{u}_j) \big| \boldsymbol{u} \in \mathcal{N}_j(\boldsymbol{i})\}\} \Big| j \in [k]\right)\right) \tag{59}$$

Based on δ-k-WL, we can easily extend $\mathcal{A}_k^{\mathsf{Ngbh}}$ to $\mathcal{A}_k^{\mathsf{Ngbh}+}$ by incorporating additional structural information $\text{adj}(\boldsymbol{i}_j, \boldsymbol{u}_j)$ via attention reweighting or attention bias, see Appendix E.1 for more details. Our $\mathcal{A}_k^{\mathsf{Ngbh}+}$ can be

regarded as the attention version of δ-k-WL and they are equivalent in expressive power, while both are strictly more expressive than k-WL and $\mathcal{A}_k^{\mathsf{Ngbh}}$. Our experiments on both synthetic datasets and real-world datasets verify the advantage of $\mathcal{A}_k^{\mathsf{Ngbh}+}$ over $\mathcal{A}_k^{\mathsf{Ngbh}}$: the former universally reveal better performance without increasing much computation cost: both $\mathcal{A}_k^{\mathsf{Ngbh}}$ and $\mathcal{A}_k^{\mathsf{Ngbh}+}$ have $O(n^{k+1})$ complexity regarding n.

Now we continue to explain our local neighbor attention mechanism. Based on δ-k-WL, Morris et al. (2020) further proposed a local variant of WL, namely δ-k-LWL, which defines the j-th *local neighbor* of tuple $\boldsymbol{i}$:

$$\mathcal{N}_j^{Local}(\boldsymbol{i}) = \{\{\psi_j(\boldsymbol{i}, v) | v \in N(\boldsymbol{i}_j)\}\} \tag{60}$$

where $\psi_j(\boldsymbol{i}, v)$ means replacing the j-th element in $\boldsymbol{i}$ with v, and $N(\boldsymbol{i}_j)$ refers to the (one-dimensional) neighbors of node $\boldsymbol{i}_j$. δ-k-LWL is defined as

$$\delta\text{-}k\text{-LWL}: \boldsymbol{C}_{\boldsymbol{i}}^{t+1} = \text{hash}\left(\boldsymbol{C}_{\boldsymbol{i}}^t, \left(\{\{\boldsymbol{C}_{\boldsymbol{j}}^t \big| \boldsymbol{j} \in \mathcal{N}_j^{Local}(\boldsymbol{i})\}\} \Big| j \in [k]\right)\right) \tag{61}$$

Morris et al. (2020) showed that δ-k-WL is at least as powerful than δ-k-LWL (the gap would be a histogram of colors of full k-neighbor, as δ-k-WL is exactly as powerful as δ-k-LWL+, while the latter incorporates histogram of k-neighbor colors compared with δ-k-LWL. Interested readers please refer to Morris et al. (2020) for more details). While δ-k-WL is strictly more powerful than k-WL, the relation between δ-k-LWL and k-WL is unknown due to the aforementioned gap. However, a significant advantage of δ-k-LWL over k-WL is its lower complexity due to the sparsity of local neighbors compared with (full or global) k-neighbors, reducing the complexity from $O(n^{k+1})k$ to $O(n^k \bar{D} k)$, where $\bar{D}$ is the average node degree in the graph.

And now we are going to show that our local neighbor attention is at least as powerful as δ-k-LWL. Recall that the (k-head) local neighbor attention $\mathcal{A}_k^{\mathsf{LN}}$ is defined as

$$\left(\mathcal{A}_k^{\mathsf{LN}}(\boldsymbol{X}, \boldsymbol{X})\right)_{\boldsymbol{i}} = \text{Concat}\left[\text{softmax}\Big((x_{\boldsymbol{i}} Q^j)\big(x_{[\psi_j(\boldsymbol{i},u)|u\in N(\boldsymbol{i}_j)]} K^j\big)^\top\Big)\big(x_{[\psi_j(\boldsymbol{i},u)|u\in N(\boldsymbol{i}_j)]} V^j\big) \Big| j \in [k]\right] \tag{62}$$

where $N(\boldsymbol{i}_j)$ denotes the set of neighbors of the j-th element in tuple $\boldsymbol{i}$, and $\psi_j(\boldsymbol{i}, u)$ still refers to replacing the j-th element in $\boldsymbol{i}$ with u.

Theorem B.6. (Theorem 4.3 in main text.) *Local neighbor attention $\mathcal{A}_k^{\mathsf{LN}}$ with residual connection, output MLPs and k heads is at least as powerful as δ-k-LWL.*

Proof. We use the same construction as in our proof for $\mathcal{A}_k^{\mathsf{Ngbh}}$. Define

$$x_{\boldsymbol{i}} Q^j = \vec{1} \in \mathbb{R}^{d_k}, \boldsymbol{i} \in \mathbb{R}^{n^k}, j \in [k] \tag{63}$$

$$x_{\boldsymbol{i}} K^j = \vec{1} \in \mathbb{R}^{d_k}, \boldsymbol{i} \in \mathbb{R}^{n^k}, j \in [k] \tag{64}$$

$$\tag{65}$$

where d_k is still the constant latent dimension. Then the local neighbor attention reduces to

$$\left(\mathcal{A}_k^{\mathsf{LN}}(\boldsymbol{X}, \boldsymbol{X})\right)_{\boldsymbol{i}} = \text{Concat}\left[\frac{1}{d(\boldsymbol{i}_j)} \sum_{u\in N(\boldsymbol{i}_j)} x_{\psi_j(\boldsymbol{i},u)} V^j \Big| j \in [k]\right] \tag{66}$$

where $d(\boldsymbol{i}_j)$ is the degree of node $\boldsymbol{i}_j$. This is an instance of δ-k-LGNN (Morris et al., 2020) with mean aggregation, which has the same expressive power as δ-k-LWL (Morris et al., 2020).

We can also provide proof for local neighbor attention with slight modifications under our theoretical framework as in (Maron et al., 2019). According to Remark 5, if we want to keep the representation of multisets injective, a summation instead of the "averaging" operation is needed, i.e. we want to eliminate the $d(\boldsymbol{i}_j)$ factor:

$$\left(\mathcal{A}_k^{\mathsf{LN}}(\boldsymbol{X}, \boldsymbol{X})\right)_{\boldsymbol{i}} = \text{Concat}\left[\sum_{u\in N(\boldsymbol{i}_j)} x_{\psi_j(\boldsymbol{i},u)} V^j \Big| j \in [k]\right] \tag{67}$$

Several solutions are applicable to obatin Equation (67): (1) replace the softmax in local neighbor attention with element-wise relu activation, so that the local neighbor attention changes to summation over tuples in each local neighbor; (2) attach node degrees as part of the input, so that the output MLPs can multiply $d(\boldsymbol{i}_j)$ back to recover the summation operation. Hence, the local neighbor attention exactly aggregate information from tuples in j-th local neighbor in the j-th head, for $j \in [k]$ respectively. Then according to Remark 3 and Remark 5, the concat operation, residual connection, and output MLPs completely simulate δ-k-LWL in Equation (61) with proper constructions.

□

A direct corollary is that $\mathcal{A}_k^{\mathsf{LN}}$ is strictly less expressive than $\mathcal{A}_k^{\mathsf{Ngbh}+}$, since δ-k-LWL is strictly less expressive than δ-k-WL (while the latter is strictly more expressive than k-WL). However, the relation between $\mathcal{A}_k^{\mathsf{LN}}$ and k-WL is unknown: there are some non-isomorphic graph pairs that $\mathcal{A}_k^{\mathsf{LN}}$ can distinguish but k-WL fails (see the CSL experiment in our main text), while the other direction is still an open question.

Virtual Tuple Attention. Finally we provide proofs for the virtual tuple attention mechanism. Recall the definition of virtual tuple attention $\mathcal{A}_k^{\mathsf{VT}}$ is

$$\mathcal{A}_k^{\mathsf{VT}}(\boldsymbol{X}', \boldsymbol{X}')_{n^k+1} = \mathrm{softmax}\Big((x'Q^1)(\boldsymbol{X}K^1)^\top\Big)\boldsymbol{X}V^1 \tag{68}$$

$$\mathcal{A}_k^{\mathsf{VT}}(\boldsymbol{X}', \boldsymbol{X}')_{\boldsymbol{i}} = x'V^2 \tag{69}$$

where the feature of virtual tuple is denoted as x', and the input is augmented to $\boldsymbol{X}' = [\boldsymbol{X}, x'] \in \mathbb{R}^{(n^k+1)\times d}$.

Proposition B.7. (Proposition 4.4 in main text.) *$O(1)$ depth and $O(1)$ width virtual tuple attention $\mathcal{A}_k^{\mathsf{VT}}$ can approximate $\mathcal{A}_k$ with Performer or Linear-Transformer architecture arbitrarily well.*

Proof. As stated before, all tuples are treated as one single equivalence class in $\mathcal{A}_k$, its calculation in the Performer or Linear-Transformer architecture is the same as the standard transformer $\mathcal{A}_1$ except for the number of tuples. Therefore, $\mathcal{A}_k$ with kernel tricks can be directly reformed to the normal Performer or Linear-Transformer with n^k inputs from the same equivalence class. Hence the $O(1)$ depth and $O(1)$ width simplified MPNN + virtual tuple (Cai et al., 2023) operating on n^k input tokens can approximate $\mathcal{A}_k$ with Performer or Linear-Transformer architecture arbitrarily well. As in (Cai et al., 2023), here 'simplified' suggests that we ignore message passing within real node neighbors, and the update function is parameterized with heterogeneous parameters for virtual tuples and real tuples. However, all real tuples are treated as the same equivalence class.

Therefore, we only need to show that virtual tuple attention can recover simplified message passing neural networks (operating on homogeneous k-tuples) + virtual tuple. To this end, we simply define

$$x'Q^1 = 1, \boldsymbol{X}K^1 = \vec{1}_{n^k} \tag{70}$$

The virtual tuple attention reduces to

$$\mathcal{A}_k^{\mathsf{VT}}(\boldsymbol{X}', \boldsymbol{X}')_{n^k+1} = \frac{1}{n^k}\sum_{i=1}^{n^k} \boldsymbol{X}_i V^1 \tag{71}$$

$$\mathcal{A}_k^{\mathsf{VT}}(\boldsymbol{X}', \boldsymbol{X}')_{\boldsymbol{i}} = x'V^2 \tag{72}$$

When the above update function is augmented with input and output MLPs, it is exactly reduced to the update function of the simplified MPNN + virtual tuple with mean aggregation function (Cai et al., 2023) which treats all tuples as one equivalence class.

Denote input features as $\boldsymbol{X}^{(0)}$, the kernelized $\mathcal{A}_k$ computes (the same as in Equation (52))

$$\boldsymbol{X}_i^{(new)} = \frac{\Big(\phi(\boldsymbol{X}_i^{(0)}Q)^\top \sum_{j=1}^{n^k}\big(\phi(\boldsymbol{X}_j^{(0)}K)\otimes(\boldsymbol{X}_j^{(0)}V)\big)\Big)^\top}{\phi(\boldsymbol{X}_i^{(0)}Q)^\top \sum_{l=1}^{n^k}\phi(\boldsymbol{X}_l^{(0)}K)} \tag{73}$$

Under the compactness assumption of the inputs and weight matrices (stated in Appendix B.1.1), we consider the following construction of two layers of virtual tuple attention (along with the residual connection and MLP). Intuitively, in the first layer (denoted by (1)), we first process each real node with an input MLP θ to compute $\theta(\boldsymbol{X}_j^{(0)}) := \text{ReshapeTo1D}\big(\phi(\boldsymbol{X}_j^{(0)}K) \otimes (\boldsymbol{X}_j^{(0)}V)\big)$; then through the first attention computation, the virtual tuple has the feature $x'^{(1)} = \frac{1}{n^k}\sum_{i=1}^{n^k}[\boldsymbol{X}_i^{(0)}, \theta(\boldsymbol{X}_i^{(0)})]V^{1(1)}$, where $[\cdot,\cdot]$ means concatenation, and the the input dimension for $V^{1(1)}$ would be $(m+1)d$ instead of d. We will show that along with the output MLPs $\psi^{(1)}$ and the feature $x'^{(1)}$, the virtual tuple can approximate $\sum_{j=1}^{n^k}\big(\phi(\boldsymbol{X}_j^{(0)}K) \otimes (\boldsymbol{X}_j^{(0)}V)\big)$ and $\sum_{l=1}^{n^k}\phi(\boldsymbol{X}_l^{(0)}K)$. Then in the second layer denoted by (2), the virtual tuple sends the message back to each real tuple, and each real tuple $\boldsymbol{i}$ can approximate Equation (52) along with the feature $x'^{(1)}V^{2(2)}$, $\boldsymbol{X}_{\boldsymbol{i}}^{(0)}$ (via residual connection) and the output MLPs $\psi^{(2)}$.

In detail, in the first layer along with the MLPs $\psi^{(1)}$, the virtual tuple updates as follows,

$$x'^{(1)} := \psi^{(1)}\Big(\frac{1}{n^k}\sum_{i=1}^{n^k}[\boldsymbol{X}_i^{(0)}, \theta(\boldsymbol{X}_i^{(0)})]V^{1(1)}\Big) = \Big[\sum_{l=1}^{n^k}\phi(\boldsymbol{X}_l^{(0)}K), \text{ReshapeTo1D}\big(\sum_{j=1}^{n^k}\phi(\boldsymbol{X}_j^{(0)}K) \otimes (\boldsymbol{X}_j^{(0)}V)\big)\Big] \tag{74}$$

where $\phi(\boldsymbol{X}_j^{(0)}K) \otimes (\boldsymbol{X}_j^{(0)}V) \in \mathbb{R}^{md'}$ is reshaped into 1-dimensional feature vector by ReshapeTo1D in raster order. Therefore, the final dimension of the virtual tuple feature $x'^{(1)}$ is $m(d'+1)$.

In the second layer, each real tuple $\boldsymbol{i}$ receives information $x'^{(1)}$ from the virtual tuple. The residual connection reserve the input feature $\boldsymbol{X}_{\boldsymbol{i}}^{(0)}$, then with MLPs $\psi^{(2)}$

$$\boldsymbol{X}_{\boldsymbol{i}}^{(2)} := \psi^{(2)}(\boldsymbol{X}_{\boldsymbol{i}}^{(0)}, x'^{(1)}V^{2(2)}) = \frac{\Big(\phi(\boldsymbol{X}_i^{(0)}Q)^\top \sum_{j=1}^{n^k}\big(\phi(\boldsymbol{X}_j^{(0)}K) \otimes (\boldsymbol{X}_j^{(0)}V)\big)\Big)^\top}{\phi(\boldsymbol{X}_i^{(0)}Q)^\top \sum_{l=1}^{n^k}\phi(\boldsymbol{X}_l^{(0)}K)} \tag{75}$$

We need to show that the above equations can be approximated arbitrarily well by MLPs $\psi^{(1)}, \psi^{(2)}$ with $O(1)$ width and $O(1)$ depth. By the uniform continuity of the functions, it suffices to show that 1) we can approxiate ϕ, 2) we can approximate multiplication and vector-scalar division, 3) the denominator $\phi(\boldsymbol{X}_i^{(0)}Q)^\top \sum_{l=1}^{n^k}\phi(\boldsymbol{X}_l^{(0)}K)$ is uniformly lower bounded by a positive number for any node features.

For 1), each component of ϕ (in both Performer and Linear Transformer)is continuous, and all inputs $\boldsymbol{X}_j^{(0)}Q, \boldsymbol{X}_j^{(0)}K$ lie in compact domain. Therefore, ϕ can be approximated arbitrarily well by MLP with $O(1)$ width and $O(1)$ depth (Cybenko, 1989).

For 2), since multiplication and vector-scalar division operations are all continuous, it suffices to show that all operands lie in a compact domain. This is true since $\boldsymbol{X}^{(0)}$ and Q, K, V are all compact, ϕ is continuous, and n is fixed. Lastly, since all these operations do not involve n, the depth and width of the MLPs are constant in n.

For 3), we need to show that the denominator is bound by a positive constant. In Performer, $\phi(\boldsymbol{X}) = \frac{\exp\left(-\frac{||X||_2^2}{2}\right)}{\sqrt{m}}[\exp(\mathbf{w}_1^T\boldsymbol{X}), \ldots, \exp(\mathbf{w}_m^T\boldsymbol{X})]$ where $\mathbf{w}_k \sim \mathcal{N}(0, I_d)$. As $||\mathbf{w}_i^T\boldsymbol{X}|| \leq ||\mathbf{w}_i|| \cdot ||\boldsymbol{X}||$, which implies that $\exp(\mathbf{w}_i^T\boldsymbol{X})$ is lower bounded by $\exp(-||\mathbf{w}_i|| \cdot ||\boldsymbol{X}||)$. Consequently, the demonimator $\phi(\boldsymbol{X}_i^{(0)}Q)^\top \sum_{l=1}^{n^k}\phi(\boldsymbol{X}_l^{(0)}K)$ is lower bounded.

For Linear Transformer, the proof is essentially the same as Performer. It boils down to showing that $\phi(\mathbf{x}) = elu(\mathbf{x}) + 1$ is continuous and positive, which is indeed the case.

□

C SIMPLICIAL TRANSFORMERS

In this section, we will detail the theoretical properties and our designs for simplicial transformers. This part provides results additional to the main text (where we do not discuss simplicial transformers in detail due to limited space), yet can be regarded as a natural extension. Simplicial complex is a different concept compared to the graph, which we will introduce in detail below. However, as discussed in the main text, k-simplices

are generally more sparse than $k+1$-tuples and are always a subset of the latter. Hence, simplices can be regarded as a result of sampling tuples, and the simplicial (complexes) transformer is a sparse variant of the tuple-based transformer we discussed in the main text. Moreover, the sparse attention mechanisms and the design principles we proposed for tuple-based transformers can be naturally extended to simplicial transformers with slight modifications, resulting in more efficient models that are still powerful.

C.1 Background of Algebraic Topology Theories

To provide readers with basic knowledge related to simplicial complexes, we introduce some existing fundamental algebraic topology theories in this subsection.

An abstract simplicial complex, denoted as $\mathcal{K}$, is defined over a finite set V, which comprises subsets of V adhering to the property of closure under inclusion. Specifically, V represents a set of vertices, denoted by $[n] = 1, 2, \ldots, n$. A subset within $\mathcal{K}$, having cardinality $k+1$, is termed a k-simplex. To illustrate, vertices are 0-simplices, directed edges are 1-simplices, and oriented triangles or 3-cliques are 2-simplices. The set of all k-simplices in $\mathcal{K}$ is represented as $S_k(\mathcal{K})$. A k-simplices has a dimension of k, and the dimension of the complex $\mathcal{K}$ itself is the maximum dimension among all its faces.

When two $(k+1)$-simplices share a common k-face (a subset of a simplex), they are termed as k-down neighbors. Conversely, two k-simplices that share a $(k+1)$-simplex are known as $(k+1)$-up neighbors. Furthermore, a k-cochain or k-form is a function defined on $\mathcal{K}_{k+1}$, $f : V \times \cdots \times V \to \mathbb{R}$ that is equivariant to the permutation. Although k-cochains have the structure of vector spaces, they are usually called cochain groups $\mathcal{C}^k(\mathcal{K}, \mathbb{R})$. Chain groups $\mathcal{C}_k(\mathcal{K}, \mathbb{R})$ are defined as duals of co-chain groups.

The simplicial coboundary maps $\delta_k : \mathcal{C}^k(\mathcal{K}, \mathbb{R}) \to \mathcal{C}^{k+1}(\mathcal{K}, \mathbb{R})$ is defined as

$$(\delta_k f)([v_0, \ldots, v_{i+1}]) = \sum_{j=0}^{k+1} (-1)^j f([v_0, \ldots, \hat{V}^j, \ldots, v_{k+1}]) \tag{76}$$

where $\hat{V}^j$ suggests that the vertex v_j is omitted. Further, we can define the adjoint of coboundary operator: $\delta_k^* : \mathcal{C}^{k+1}(\mathcal{K}, \mathbb{R}) \to \mathcal{C}^k(\mathcal{K}, \mathbb{R})$.

Utilizing the concept of boundary and coboundary, the Hodge k-Laplacian operator (also called the combinatorial Laplace operator) is defined as:

$$\mathbf{L}_k = \mathbf{L}_{k,down} + \mathbf{L}_{k,up} = \delta_{k-1}\delta_{k-1}^* + \delta_k^*\delta_k \tag{77}$$

where we omit the reference to simplicial complex $\mathcal{K}$ from the notation for simplicity. By definition, all three operators $\mathbf{L}_k, \mathbf{L}_{k,up}, \mathbf{L}_{k,down}$ are self-adjoint, nonnegative and compact.

In the Hilbert space, the matrix representation for boundary and co-boundary operators are equivalent to adjacent matrix of k and $k+1$ order simplices. We write the matrix representation for δ_k^* as $\mathbf{B}_{k+1} \in \mathbb{R}^{|S_k| \times |S_{k+1}|}$ (one can view it as the adjacent matrix of k-th and $k+1$-th simplices). Therefore, in this paper we use the following definition for Hodge Laplacians:

$$\mathbf{L}_k = \mathbf{B}_k^*\mathbf{B}_k + \mathbf{B}_{k+1}\mathbf{B}_{k+1}^* \tag{78}$$

where $\mathbf{B}_k^* = \mathbf{B}_k^T$ is the adjoint of $\mathbf{B}_k$, which is equivalent to the transpose of $\mathbf{B}_k$ in the Hilbert space. Specifically, when $k = 0$, $\mathbf{L}_0$ is exactly the graph Laplacian $\mathbf{L}_0 = \mathbf{D} - \mathbf{A}$.

Furthermore, the Hodge Laplacian of the entire K-dimensional simplicial complex $\mathcal{K}$ is a block diagonal matrix $\mathbf{L}(\mathcal{K})$, with the k-th block being $\mathbf{L}_k(\mathcal{K})$ for $k = 0, \ldots, K$. If δ is the exterior derivative of a finite abstract simplicial complex $\mathcal{K}$, then

$$\mathbf{L}(\mathcal{K}) = \mathbf{D}^2 = (\delta + \delta^*)^2 = \delta\delta^* + \delta^*\delta \tag{79}$$

where $\mathbf{D} = \delta + \delta^*$ is the Dirac matrix.

Zhou et al. (2023a) further defines the inter-order Hodge Laplacian for a K-order simplicial complex $\mathcal{K}$, denoted as $\mathcal{L}_{0:K}(\mathcal{K})$. This generalized inter-order Hodge Laplacian contains information (Hodge Laplacians, boundaries, coboundaries) for all $k \in [K]$, which is defined as:

$$\mathcal{L}_{0:K}(\mathcal{K}) = \begin{bmatrix} \mathbf{L}_0 & \mathbf{B}_1 & & & & \\ \mathbf{B}_1^T & \mathbf{L}_1 & \mathbf{B}_2 & & & \\ & \cdots & \cdots & \cdots & & \\ & & \cdots & \cdots & \cdots & \\ & & & \mathbf{B}_{K-1}^T & \mathbf{L}_{K-1} & \mathbf{B}_K \\ & & & & \mathbf{B}_K^T & \mathbf{L}_K \end{bmatrix} \tag{80}$$

$\mathcal{L}_{0:K}(\mathcal{K})$ contains information for all $k \leq K$ order simplices, which is a block matrix with $\mathbf{L}_k$ in the k-th diagonal block, $\mathbf{B}_k^T$ and $\mathbf{B}_{k+1}$ in the offset ± 1 diagonal blocks, while all other blocks are zeros.

As discussed in (Zhou et al., 2023a), a number of previous works such as (Bodnar et al., 2021b) can be reformatted and unified by $\mathcal{L}_{0:K}$. We will show that simplicial transformer encoding $\mathcal{L}_{0:K}$ as attention bias can generalize simplicial convolutions in the next subsection.

In addition, we can make use of $\mathcal{L}_{0:K}^r$ to build random walk-based positional encoding for all simplices in the K-dimensional simplicial complex that contains more information than random walks within the same order simplices. Similar to (Bodnar et al., 2021a), we can also introduce any form of local structure (such as rings and cycles) as expanded complex cells and perform random walk on them, which can greatly facilitate graph learning by incorporating higher order structures other than simplicial complexes (nodes, edges, triangles, and four-cliques).

C.2 Simplicial Transformer with Global Attention

As discussed in Appendix A, although there are some networks defined on simplices and simplicial complexes, most of them are in convolutional or message passing manners (Bodnar et al., 2021b; Yang and Isufi, 2023). Other attention-based models all restrict their attention reception field within the range of boundaries, co-boundaries, upper and lower adjacent neighbors (Goh et al., 2022; Giusti et al., 2022), which can be viewed as weighted convolutional simplicial networks.

In this subsection, we propose our novel full simplicial transformer, theoretically analyze the pros and cons brought by global reception field, and systematically discuss the design spaces of simplicial transformers.

Full Simplicial Transformer with Global Attention. The simplicial transformer with global attention is defined as follows.

Definition C.1. Denote k-simplices of an abstract simplicial complex $\mathcal{K}$ as $S_k(\mathcal{K})$, whose corresponding features are $\boldsymbol{X} \in \mathbb{R}^{|S_k(\mathcal{K})| \times d}$. The k-dimensional simplicial self-attention is defined as $\mathcal{AS}_k$:

$$\mathcal{AS}_k(S_k(\mathcal{K})) = \text{softmax}\Big(\boldsymbol{X}Q(\boldsymbol{X}K)^\top\Big)\boldsymbol{X}V \tag{81}$$

where we use the notation $\mathcal{AS}_k$ to differentiate the transformer defined on k-simplices from the one defined on k-tuples in our previous sections. For simplicity, both the k-dimensional simplicial self-attention and the simplicial transformer (i.e. self-attention along with components such as residual connections and output MLPs) are denoted as $\mathcal{AS}_k$ due to their mild difference.

The above $\mathcal{AS}_k$ involves only k-simplices, yet it is natural to extend the input tokens to all simplices with order $k \leq K$ for a K-dimensional simplicial complex $\mathcal{K}$.

Definition C.2. Denote the set of all k-simplices of a K-dimensional abstract simplicial complex $\mathcal{K}$ where $k \leq K$ as $S_{0:K}(\mathcal{K})$, whose corresponding features are $\boldsymbol{X} \in \mathbb{R}^{\sum_{k=0}^K |S_k(\mathcal{K})| \times d}$. The $0:K$-dimensional simplicial self-attention is defined as $\mathcal{AS}_{0:K}$:

$$\mathcal{AS}_{0:K}(S_{0:K}(\mathcal{K})) = \text{softmax}\Big(\boldsymbol{X}Q(\boldsymbol{X}K)^\top\Big)\boldsymbol{X}V \tag{82}$$

Hodge Laplacians as Attention Bias. Similar to the lost of connectivity information in standard graph transformer due to dense attention, one may observe that the dense simplicial transformer $\mathcal{AS}_k$ and $\mathcal{AS}_{0:K}$ are also unaware of the connectivity and structure information, including coboundaries, boundaries as well as upper and lower adjacent neighbors. To address this problem of standard (first order) graph transformer, a variety of

models are proposed to encode structure information via different approaches, among which Graphormer (Ying et al., 2021) is a well known model. Concretely, Graphormer embeds node degrees, edge features and pair-wise shortest path distances to the $n \times n$ attention matrix. In its first order case, node degree and edge information can be summarized into the standard graph Laplacian L_0.

Now we generalize this design to the arbitrary-order simplicial transformer. For k-simplices, Hodge k Laplacian $\mathbf{L}_k$ summarizes the upper adjacent and lower adjacent information into a $|S_k(\mathcal{K})| \times |S_k(\mathcal{K})|$ matrix, where $S_k(\mathcal{K})$ is the number of k-simplices in $\mathcal{K}$. Therefore, $\mathcal{AS}_k$ and $\mathcal{AS}_{0:K}$ with attention biases are defined as:

$$\mathcal{AS}_k(S_k(\mathcal{K})) = \text{softmax}\Big(\boldsymbol{X}_k Q(\boldsymbol{X}_k K)^\top + \phi(\mathbf{L}_k)\Big)\boldsymbol{X}_k V \tag{83}$$

$$\mathcal{AS}_{0:K}(S_k(\mathcal{K})) = \text{softmax}\Big(\boldsymbol{X} Q(\boldsymbol{X} K)^\top + \phi(\mathbf{L})\Big)\Big(\text{Concat}\big[(\boldsymbol{X}_0 V^0)^\top, (\boldsymbol{X}_1 V^1)^\top, \ldots, (\boldsymbol{X}_K V^K)^\top\big]\Big)^\top,\ k = 0, \ldots, K \tag{84}$$

where $\boldsymbol{X}_k \in \mathbb{R}^{|S_k(\mathcal{K})| \times d}$ is the feature of $S_k(\mathcal{K})$ (i.e. the k-faces), $\boldsymbol{X} \in \mathbb{R}^{\sum_{k=0}^{K} |S_k(\mathcal{K})| \times d}$ is the concatenation of $\boldsymbol{X}_k$, $\mathbf{L}$ is the Hodge Laplacian of $\mathcal{K}$, and $\mathbf{L}_k$ is the k-th order Hodge Laplacian; ϕ is an element-wise function, e.g. an identity function, an element-wise MLP etc.

In addition, we also introduce an augmented Hodge Laplacian $\mathcal{L}_{0:K}$, see Appendix C.1 for more details. The key difference between $\mathcal{L}_{0:K}$ and the standard Hodge Laplacian $\mathbf{L}$ is that the former has co-boundary and boundary operators in the ± 1 off-diagonal blocks, which are zeros in the latter.

Furthermore, by reweighting the attention using the Hodge Laplacians, we can recover the simplicial message passing networks. In particular,

$$\mathcal{AS}_{0:K}(S_k(\mathcal{K})) = \Big(\text{softmax}\big(\boldsymbol{X} Q(\boldsymbol{X} K)^\top\big) \odot \phi(\mathbf{L})\Big)\Big(\text{Concat}\big[(\boldsymbol{X}_0 V^0)^\top, (\boldsymbol{X}_1 V^1)^\top, \ldots, (\boldsymbol{X}_K V^K)^\top\big]\Big)^\top,\ k = 0, \ldots, K \tag{85}$$

where $\odot$ is the Hadamard product (element-wise product). We will show the benefit of including Hodge-k Laplacian in simplicial transformers in the following subsection, making connections between simplicial transformers and simplicial message passing networks, $\mathcal{A}_k$ defined on k-tuples as well as k-WL hierarchy. For each order k, the diagonal block $\mathbf{L}_k$ in $\mathbf{L}$ enable information aggregation from upper and lower adjacent neighbors, which are also k-simplices. However, the message cannot be passed among simplices of different orders through boundaries and coboundaries through $\mathbf{L}$, and $\mathcal{L}_{0:K}$ addresses this problem by introducing boundary and coboundary operators into off-diagonal blocks.

C.3 Theoretical Analysis on Simplicial Transformers

In this subsection, we present our theoretical results concerning simplicial transformers. Concretely, we establish the connections between our proposed simplicial transformers and two existing families of models: simplicial message passing networks and k-IGN (k-WL). We also put forward a spectral monotonicity result of the attention matrix of the simpilcial transformer, which gives more insights into the relationship between simplicial complex-based models and tuple-based models.

Simplicial Transformer Generalizes Simplicial Message Passing Networks. We show that our simplicial transformer with global attention and Hodge Laplacian as attention bias is a more general version of simplicial networks in message-passing or convolution manners.

Theorem C.3. *Simplicial transformers $\mathcal{AS}_{0:K}$ reweighted by augmented Hodge Laplacian $\mathcal{L}_{0:K}$ encodings (in Equation (85)) can approximate message passing and convolutional simplicial networks.*

Proof. Bodnar et al. (2021b) already proved that their Message Passing Simplicial Network and WL variant SWL generalizes convolutional simplicial networks such as (Ebli et al., 2020). Now we only need to show that $\mathcal{AS}_{0:K}$ with $\mathcal{L}_{0:K}$ encodings as attention bias can exactly simulate the Message Passing Simplicial Network. WLOG, we first consider the update function of k-simplices ($0 < k < K$). Let $\boldsymbol{X}_k \in \mathbb{R}^{|S_k(\mathcal{K})| \times d}$ represent the feature of all k-faces, we construct

$$\boldsymbol{X}_j Q = 1\vec{1}_{|S_j|}, \boldsymbol{X}_j K = \vec{1}_{|S_j|},\ j = 0, \ldots, K \tag{86}$$

We define the element-wise function as $\phi(\cdot) = \cdot \times \mathbb{1}(\cdot)$, where $\mathbb{1}(\cdot) = 1$ if the element is in the non-zero block of $\mathcal{L}_{0:K}$ and 0 otherwise. Define $V^k = W_k \times (\sum_{k=0}^{K} |S - k|)$, the update function can becomes,

$$\mathcal{AS}_{0:K}(\boldsymbol{X}_k) \to \delta_{k-1}\boldsymbol{X}_{k-1}W_{k-1} + \mathbf{L}_k\boldsymbol{X}_kW_k + \delta_k^*\boldsymbol{X}_{k+1}W_{k+1} \tag{87}$$

According to the definition of $\delta_{k-1}, \mathbf{L}_k, \delta_k^*$, the three terms on the right recover the message from the boundaries, the lower and upper adjacent neighbors, and the co-boundaries, respectively. This is exactly the update function of MPSN, see Equations (2)-(6) in (Bodnar et al., 2021b). Therefore, our $\mathcal{AS}_{0:K}$ with $\mathcal{L}_{0:K}$ encodings as attention bias approximates MPSN and other convolutional simplicial networks arbitrarily well.

□

While our sipmlicial transformer with attention bias has the capability to recover MPSN (Bodnar et al., 2021b) and other convolutional simplicial networks, the other direction is obviously not true due to the local reception field of the message passing scheme. Hence, taking advantage of the global nature of simplicial transformers may be a promising direction.

Spectral Monotonicity of Simplicial Attention. Now we provide some additional results on the spectral monotonicity of the simplicial attention, which helps us better understand the behaviors of our simplicial transformer given a series of monotonic simplicial complexes $\mathcal{K}_1 \subset \cdots \subset \mathcal{K}_N$, thus establishing connections with transformers on tuples.

To start with, we consider two simplicial complexes $\mathcal{K}_1$ and $\mathcal{K}_2$ with $m_1 < m_2$ elements respectively, where $\mathcal{K}_1$ is a sub-simplicial complex of $\mathcal{K}_2$. To make their spectra comparable, define $\lambda_i(\mathcal{K}_1) = 0$ for $i \leq m_2 - m_1$ and $\lambda_{m_2-m_1+i}(\mathcal{K}_1) = \mu_i(\mathcal{K}_1)$, where μ_i are the original m_1 eigenvalues of Hodge Laplacian $\mathcal{L}(\mathcal{K}_1)$ ordered in ascending order. Together, spectra of two simplicial complexes can be seen as left-padded non-descending sequences, thus comparable.

We first introduce a known result on spectral monotonicity of Hodge Laplacian.

Lemma C.4.

$$\lambda_j(\mathcal{K}_1) \leq \lambda_j(\mathcal{K}_2), \forall j \leq m_2 \tag{88}$$

The original proof is given in Knill (2023). Next, we give a novel spectral monotonicity result on the attention matrix of a simplicial transformer.

Theorem C.5. *Suppose $\mathcal{K}_1 \subset \mathcal{K}_2$, consider a simplicial transformer $\mathcal{AS}_{0:K}$ that uses the Hodge Laplacian $\mathbf{L}(\mathcal{K})$ as attention bias. Suppose the following mild conditions hold: (i) projection matrix $Q = K$, which guarantees the symmetry of attention matrix; (ii) $\boldsymbol{X}Q$ has all elements positive, which can be easily achieved via a ReLU activation. Denote the attention matrices of $\mathcal{K}_1$ and $\mathcal{K}_2$ (i.e. the $m_i \times m_i$ matrices before placed in softmax, $i = 1, 2$) as $\boldsymbol{A}_i = (\boldsymbol{X}_iQ)(\boldsymbol{X}_iQ)^\top + \mathbf{L}(\mathcal{K}_i),\ i = 1, 2$ follow the spectral monotonicity:*

$$\lambda_j(\boldsymbol{A}_1) \leq \lambda_j(\boldsymbol{A}_2), \forall j \leq m_2 \tag{89}$$

Proof. The padding rule of the eigenvalues is the same as we stated. If necessary, the feature $\boldsymbol{X}Q$ is allowed to be *zero-padded* for the simplices in $\mathcal{K}_2$ but not in $\mathcal{K}_1$, thus $\boldsymbol{X}_1$ is padded to the same dimension as $\boldsymbol{X}_2$. Our proof parallels those of Kirchhoff Laplacian L_0 (Spielman, 2009) and Hodge Laplacian (Knill, 2023). Suppose G is a finite set of non-empty sets closed under the operation of taking finite non-empty subsets, a set $x \in G$ is then called *locally maximal* if it is not contained in an other simplex. This suggests that the set $U = \{x\}$ is an open set in the non-Hausdorff Alexandroff topology $\mathcal{O}$ on G generated by the basis formed by $U(x) = \{y \in G, x \subset y\}$. In this case the spectrum changes monotonically if we add a locally maximal simplex to a given complex (Knill, 2023). Now, since $\mathcal{K}_1 \subset \mathcal{K}_2$, $\mathbf{L}(\mathcal{K}_1) \leq \mathbf{L}(\mathcal{K}_2)$ is in the Loewner partial order. Recalling that $\mathbf{L}(\mathcal{K}) = \mathbf{D}(\mathcal{K})^2$, the following statement always holds:

If $u := \mathcal{K}_2 \to \mathbb{R}$ is a vector, then the quadratic form of the attention matrix $\boldsymbol{A}$ is

$$\langle u, \boldsymbol{A}u\rangle = \langle u, (\boldsymbol{X}Q)(\boldsymbol{X}Q)^\top + \mathbf{L})u\rangle \tag{90}$$
$$= \langle u, (\boldsymbol{X}Q)(\boldsymbol{X}Q)^\top + \mathbf{D}^2)u\rangle \tag{91}$$
$$= \langle \mathbf{D}u, \mathbf{D}u\rangle + \langle (\boldsymbol{X}Q)^\top u, (\boldsymbol{X}Q)^\top u\rangle \tag{92}$$
$$= ||\mathbf{D}u||^2 + ||(\boldsymbol{X}Q)^\top u||^2 \tag{93}$$

According to the spectral monotonicity of Hodge Laplacians in Theorem C.4, we already have that $||\mathbf{D}_1 u||^2 \leq ||\mathbf{D}_2 u||^2$ always holds for any vector u. We aim to show that the second quadratic form also always increases when adding new maximal simplices. When adding the $m_2 - m_1$ components that exist only in $\mathcal{K}_2$, we have

$$||(\boldsymbol{X}_2 Q)^\top u||^2 - ||(\boldsymbol{X}_1 Q)^\top u||^2 = \sum_{i=m_1+1}^{m_2} ||(\boldsymbol{X}_2 Q)_i u_i||^2 \geq 0 \tag{94}$$

where we use $(\boldsymbol{X}_2 Q)_{1:m_1} = (\boldsymbol{X}_1 Q)_{1:m_1}, (\boldsymbol{X}_1 Q)_{m_1+1:m_2} = \vec{0}$, i.e. we use zero padding. Together, $\langle u, \boldsymbol{A}_1 u\rangle < \langle u, \boldsymbol{A}_2 u\rangle$ always holds, indicating that adding new maximal simplices only increases the total quadratic form. Then, *Courant Fischer Theorem* gives the result using $\mathcal{S}_k = \{V \subset \mathbb{R}^n, \dim(V) = k\}$

$$\lambda_k(\boldsymbol{A}_1) = \min_{V\in\mathcal{S}_k} \max_{|u|=1, u\in V} \langle u, \boldsymbol{A}_1 u\rangle \leq \min_{V\in\mathcal{S}_k} \max_{|u|=1, u\in V} \langle u, \boldsymbol{A}_2 u\rangle = \lambda_k(\boldsymbol{A}_2) \tag{95}$$

□

The spectral monotonicity result may give us some intuitions of the connection between simplicial transformers and tuple transformers mentioned before. As a tuple transformer always takes all tuples into account, it can be viewed as a simplicial transformer operating on all possible faces defined on $V = [n]$ (or, briefly speaking, on a complete graph), whose attention matrix always has the largest corresponding values compared to other simplicial complexes. In specific, an order-k simplicial transformer computes attention between the simplicial complexes, a subset of all k-tuples considered by an order-k tuple transformer. Consequently, their attention matrices follow the spectral monotonicity analyzed in Theorem C.5. To some extent, the computation of simplicial transformer is more sparse and stable compared with the full transformer defined on tuples, although the latter may avoid the expansion of attention matrix via breaking the two conditions of symmetric attention and positive inputs. More comparison and theoretical connections between simplicial transformers and tuple transformers are worth exploring in the future, including their theoretical expressive power and practical performance.

Connections with k-WL Hierarchy. Analogous to $\mathcal{A}_k$, a pure $\mathcal{AS}_{0:K}$ cannot effectively update its representations in the sense of distinguishing non-isomorphic graphs due to the dense and homogeneous attention mechanism. However, by incorporating Hodge-Laplacians as attention bias or taking PE/SE defined on simplicial complexes as input, the theoretical expressive power and real-world performance of simplicial transformers can both be boosted.

It is still an open question to establish complete connection between simplicial networks (either convolutional or attention-based) and k-WL hierarchy. However, we can give some primary results to show the benefit of using sparse k-simplices instead of all k-tuples.

Theorem C.6. *$\mathcal{AS}_{0:3}$ with $\mathcal{L}_{0:3}$ as attention bias can distinguish a pair of non-isomorphic strongly regular graph, namely Rook's 4×4 graph and the Shrikhande graph, which cannot be distinguished by 3-WL.*

Proof. We have already shown that $\mathcal{AS}_{0:3}$ approximates the message passing simplicial network (MPSN) with order 3 arbitrarily well. Bodnar et al. (2021b) prove that MPSN of order 3 can distinguish Rook's 4×4 graph and the Shrikhande graph. It is a well-known fact that 3-WL cannot distinguish strongly regular graphs. □

C.4 Sparse Simplicial Transformers

Analogously to tuple-based transformers, our simplicial transformers also benefit from various techniques including sparse attention mechanisms, thus improving both expressive power and real-world performance.

The sparse attention mechanisms we proposed for tuple-based high order transformers can be naturally extended to simplicial transformers with only a minor modifications, where each token now becomes simplices (either of same order or of different orders, e.g. from 0-simplices to K-simplices) instead of k-tuples. For neighbor attention, the concept of (local) k-neighbor for tuples now switches to a broader definition for simplices. For a k-order simplex , we consider its coboundaries (which are $k+1$-simplices), boundaries (which are $k-1$-simplices) and upper/lower adjacent simplices (which are k-simplices) as its extended 'neighbor', see (Bodnar et al., 2021b) for more details. We denote the *simplex neighbor attention* implementation as $\mathcal{AS}^{\mathsf{SN}}_{k_1:k_2}$, where k_1 and k_2 are the lowest and highest order of the simplices we consider. Simplex neighbor attention is sparse and capable of capturing local structures. To improve its expressiveness, different types of relations (including coboundary, boundary and upper/lower adjacent, depending on the position in Hodge Laplacian $\mathcal{L}_{k_1:k_2}$) are embedded to re-weight the attention matrix. As for virtual tuple attention, it is now naturally converted to virtual simplex attention, which we denote as $\mathcal{AS}^{\mathsf{VS}}_{k_1:k_2}$. The advantage of virtual simplex attention is that it can capture global information, similar to the virtual node in MPNNs. However, it may also suffer from over-smoothing or over-squashing.

D DISCUSSION

D.1 Further Discussion on Related Work

In Appendix A we already introduce high-order transformers in (Kim et al., 2022; Sanford et al., 2023). In this subsection, we give some in-depth discussion on related works, mainly (Sanford et al., 2023).

Sanford et al. (2023) proposed another family of high-order transformers, see Appendix A for descriptions. Besides the theoretical expressive power compared with k-WL and k-FWL hierarchy mentioned in our main text, we now provide additional results from the perspective of the representation power, and particularly the ability to solve Match-m problem.

Match-m Problem. Sanford et al. (2023) presents problems of *pair detection* (Match2) and *triple detection* (Match3), which are defined for inputs $\boldsymbol{X} = (x_1, \ldots, x_n) \in [M]^{n\times d}$ (for some $M = \text{poly}(n)$) as

$$\text{Match2}(\boldsymbol{X})_{i\in[n]} = \mathbb{1}\Big(\exists j \;\; \text{s.t.} \;\; x_i + x_j = \vec{0} \;\; (\text{mod} M)\Big) \tag{96}$$

$$\text{Match3}(\boldsymbol{X})_{i\in[n]} = \mathbb{1}\Big(\exists j_1, j_2 \;\; \text{s.t.} \;\; x_i + x_{j_1} + x_{j_2} = \vec{0} \;\; (\text{mod} M)\Big) \tag{97}$$

Sanford et al. (2023) conclude that a single layer of standard transformer (i.e. $\mathcal{A}_1$) with input and output MLPs and an $O(d)$-dimensional embedding can efficiently compute Match2, but fails to compute Match3 unless the number of heads H or the embedding dimension d_k grows polynomially in n. However, they show that a certain "third-order tensor self-attention" (which resembles $\mathcal{A}_{1,2}$ in our formulation) can efficiently compute the Match3 problem with a single unit.

Their matching problem can be easily generalized to arbitrary order, called the Match-m problem, which is defined for $\boldsymbol{X} = (x_1, \ldots, x_n) \in [M]^{n\times d}$ (for some $M = \text{poly}(n)$) as

$$\text{Match}m(\boldsymbol{X})_{i\in[n]} = \mathbb{1}\Big(\exists j_1, \ldots, j_{m-1} \;\; \text{s.t.} \;\; x_i + x_{j_1} + \cdots + x_{j_{m-1}} = \vec{0} \;\; (\text{mod} M)\Big) \tag{98}$$

Following (Sanford et al., 2023), we allow a single blank token $x' = \vec{0}$ to be appended at the end of sequence $\boldsymbol{X} = (x_1, \ldots, x_n)$, and allow the existence of a positional encoding with $x_{i,0} = i$. Thus, the input to the attention is augmented as $\boldsymbol{X}' = (x_1, \ldots, x_n, x')$. Additionally, the input can be transformed by an element-wise MLP $\phi : \mathbb{R}^d \to \mathbb{R}^m$. We next show the ability of $\mathcal{A}^{1,k}_{1,1}$ transformer to address Match-m problem, which is a generalization of the conclusions in (Sanford et al., 2023).

Proposition D.1. *Order-*$1, m-1$ *Transformer in (Sanford et al., 2023) can efficiently solve the MATCH-m problem with* $d \times 2^{m-1} + 1$ *hidden width.*

The proof essentially follows the proof of Theorem 6 and Theorem 18 in (Sanford et al., 2023). We cuse a similar construction related to the trigonometric function, except that we need more terms.

Sanford et al. (2023) also give an augmented variant of high-order graph transformer (Definition 8 in (Sanford et al., 2023)), which determines each element of the self-attention tensor based on both its respective inner product (attention score) and on the presence of edges among the corresponding inputs (provided by an input adjacency matrix of the graph). This enhancement somewhat resembles our sparse mechanisms, both of which improve the representation power by forcing some elements in the attention to be zero. Some results on substructure counting using their edge-augmented high-order transformers.

Connection with Our Models. The definition of their general s-order order transformer (Definition 7 in (Sanford et al., 2023)) strongly resembles our cross-attention $\mathcal{A}_{1,s-1}$. The difference lies in that they reconstruct the $s-1$-order key and value tensor from the input 1-order via tensor product in every layer, while we explicitly maintain representations of the $s-1$-order tensor. Their method is a sort of hierarchical pooling (Zhou et al., 2023b) (they only maintain representations for 1-order tensor), and the reconstruction may not recover full information of the original $s-1$-order tensor. As a result, their method may be weaker in expressive power and representation power. Their high-order graph transformer (Definition 8 in (Sanford et al., 2023)) is augmented with an input adjacency matrix, allowing the attention to incorporate edge information and graph structure. Our model also has the ability via: (1) initialization based on isomorphism types of tuples, and (2) sparse attention mechanism based on (local) neighbor information. It would be an interesting future direction to theoretically investigate the strict relationship between our transformers and theirs. Empirically, there are some common inspirations that can be applied to practical models of both ours and theirs, including reweighting the attention score via edge information and calculating high-order tensor representations from first-order tensors. We also implement cross attention $\mathcal{A}_{1,2}$, which to some extent recovers their "3-order transformer", see Appendix E for more details.

D.2 Discussion on Other Theoretical Properties

Comparison Between Transformer and MPNN/WL/IGN. For standard transformer and MPNN, it is obvious that standard transformers without special design are not aware of structure information, while MPNN does via the edge information. Instead, MPNN can approximate the linear transformer with a virtual node (Cai et al., 2023).

The same holds for higher-order cases. Higher order (simplified) transformer without equivalence class basis (and the designs in (Kim et al., 2022)) is homogeneous, which means that all tokens play equal roles in updating one token. The only input of structural information is through the initialization of k-tuple representations. Instead, k-WL and k-IGN can distinguish different classes of k-tuples with the internal design of k-neighbor and equivalence class basis, respectively. Essentially, these designs are based on indices and are only dependent on the order k (or possibly the number of nodes). Each k-IGN layer alone is capable of aggregating all k-tuples' information, while k-WL does so in $[k/2]+1$ steps (and simulates one k-IGN layer in $k-1$ steps). As a consequence, without 'virtual node' k-IGN can still capture global information as a kernelized high-order transformer (e.g. k-Performer), and it has even more equivalence classes other than the sum of all k-tuples. Thus, the key difference between k-transformers and k-IGN lies in the pairwise multiplication in attention computation and the existence of equivalence class basis.

It is remarkable that deepsets for one dimension and high-order functions have internal gap: graph isormorphism problems cannot be reformed into deepsets, but can be reformatted to the problem on $k >= 2$ dimensions, yet k-IGN and k-WL cannot solve them. Therefore, k-IGN are not universal approximators on k-dimensional tensors, while transformers cannot even approximate k-IGN without input indices or equivalence class basis (dense connection lose expressivity).

Computation Complexity. 2-IGN needs n^4 spaces, or more generally, n^{2k} for equivariant linear layer from n^k to n^k; this can be verified in Equation (9a,9b) and (10a,10b) in Maron et al. (2018) . In other words, each k-tuple receives information from all n^k tuples, which highly resembles the computation in the transformer $\mathcal{A}_k$.

As a matter of fact, these two architectures have computational complexities of the same magnitude. Again, the key difference between k-transformers and k-IGN lies in the pairwise multiplication in attention computation and the existence of equivalence class basis. Attention may lead to stronger performance in real-world tasks, similar to the SOTA performance in CV and NLP; yet the equivalence class basis in k-IGN is crucial to distinguish non-isomorphism graphs. By introducing equivalence class basis (or equivalently, adjacency relations of k-tuples), $\mathcal{A}_k$ can be as power as k-IGN but cannot surpass the latter since attention does not increase expressive power.

Tensor Multiplication Ability. 2-PPGN can simulate matrix multiplication and Maron et al. (2019) actually generalize it to any order k, so k-PPGN (as powerful as k-FWL or $k+1$-WL) can implement a certain class of k-order tensor multiplication. However, this is different from the column-wise Kronecker product in (Sanford et al., 2023) and the normal matrix multiplications.

Separation Speed. An interesting thing is that the tensor multiplication ability does not necessarily increase the expressive power. As an example for order 2, Geerts (2020b) propose a walk-MPNN which is bounded by 2-PPGN and 2-FWL. Walk-MPNN allows multiple matrix (as it is for order-2 tensors) multiplication, but this does not increase its expressive power (still bounded by 2-FWL). Instead, multiple matrix multiplication only makes it distinguish graphs faster than 2-FWL. A similar phenomenon would be: Geerts (2020a) shows that (k-1) iterations of k-WL simulate one k-IGN layer. Therefore, k-IGN is faster (in the sense of distinguishing graphs or representation power, not computation efficiency) than k-WL due to its global computation (k-IGN makes use of all k-tuples, while k-WL only makes use of k-neighbors). Although equivalent in their expressive power, the different separation speeds of k-WL and k-IGN may lead to distinct practical performance, which also applies to our high-order transformers. Our sparse attention mechanisms resemble k-WL and are more 'local' than k-IGN. Particularly, one our neighbor attention layer is equivalent to one k-WL iteration, and $(k-1)$ layers of our neighbor attention is equivalent to one k-IGN layer with less complexity ($O(n^{k+1}k(k-1)$ for $(k-1)$ layers of neighbor attention v.s. $O(n^{2k})$ for one layer of k-IGN).

E EXPERIMENTS AND EMPIRICAL ANALYSIS

E.1 Model Implementation Details

Dense and Sparse Implementation. Generally, there are two principal ways to implement our attention: dense tensor multiplication and sparse attention. The former can be based on Einstein sum or other fast tensor multiplication methods, which is applicable to all the categories of our dense attention mechanisms. The latter can be efficiently implemented via torch-geometric relevant libraries. The efficiency of dense implementation can be improved via advanced techniques, yet the general memory and time consumption are internally higher than the sparse one, which is one of the main motivations of our work.

Initialization. As stated in our theoretical analysis, the initialization of tuples should depend on their isomorphism types. To implement this, we first embed node and edge features through an embedding layer. Then we concatenate the corresponding node and edge features according to the indices in the tuple in order, which implement an ordered set according to Remark 3. To keep the latent dimension consistent with the rest of the network, we apply MLPs to project the concatenated tuple features onto the common network width. We also offer an optional initialization method that simply sums up all node and edge features. Though not theoretically expressive, we observe that this implementation generally performs similarly to (or slightly weaker than) the concatenation initialization.

Pooling. We implement a comprehensive family of pooling functions for tuple transformers and simplicial transformers. For tuple transformers, (a) in node-level task, we perform k-times of hierarchical pooling, each pooling obtains a n^{t-1} tensor from the n^t tensor, eventually resulting in a tensor of size n corresponding to n nodes; (b) in edge-level task, if the order is 2, we directly extract the tuples containing the target edge; if the order is not 2, we perform pooling to obtain node representations as in (a), then calculate the target edge feature according to the two node features; (c) in graph-level task, we directly operate on all tuples and support the common max/mean/add pooling methods. For simplicial transformers, as they internally contain node (0-simplices) features and edge (1-simplices) features, the pooling only extracts the corresponding simplices. The graph-level pooling is similar to tuple transformers; additionally, we support both operating on all simplices or

only on 0-simplices.

Attention Reweighting and Attention Bias. In our simplicial transformer we have already described attention bias as an enhancement in terms of structure awareness, expressive power and practical performance. Consider the attention score the attention score $\boldsymbol{A}_{\boldsymbol{i},\boldsymbol{j}}$ calculated by inner-product of query token $\boldsymbol{i}$ and key token $\boldsymbol{j}$, both the attention reweighting and the attention bias provide additional information based on the pairwise relation of $r(\boldsymbol{i},\boldsymbol{j})$. Formally, we embed the relation with an embedding layer $\mathcal{E}$, then attention reweighting modifies the attention score to $\boldsymbol{A}_{\boldsymbol{i},\boldsymbol{j}} * \mathcal{E}(r(\boldsymbol{i},\boldsymbol{j}))$, while attention bias calculates $\boldsymbol{A}_{\boldsymbol{i},\boldsymbol{j}} + \mathcal{E}(r(\boldsymbol{i},\boldsymbol{j}))$. For instance, in (local) neighbor attention $r(\boldsymbol{i},\boldsymbol{j})$ indicates which (local) neighbor of $\boldsymbol{i}$ is $\boldsymbol{j}$ in (or not in); in $\mathcal{A}_k^{\mathsf{Ngbh}+}$, $r(\boldsymbol{i}, \psi_j(\boldsymbol{i}, u)) = \mathrm{adj}(\boldsymbol{i}_j, u)$, which provides additional structural information on the connectivity between $\boldsymbol{i}_j$ and u as δ-k-WL does; in simplicial transformer r is just the Hodge Laplacian $\mathbf{L}$ or $\mathcal{L}_{0:K}$ to indicate boundary, coboundary and adjacent information. Similar techniques are also adopted in other graph transformers (Ying et al., 2021; Ma et al., 2023b; Shirzad et al., 2023).

Positional and Structural Encodings. Extensive literature provided evidence that positional encodings (PE) and structural encodings (SE) can significantly improve the theoretical expressive power and real-world performance of graph transformers (Muller et al., 2023). Similarly, we can make use of PE and SE for high-order tuples to facilitate high-order graph transformers. The first way to incorporate PE and SE into high-order graph transformers is to concatenate existing PE and SE for nodes and edges (e.g. RWSE (Dwivedi et al., 2021) and LapPE (Kreuzer et al., 2021)) to compose PE/SE for tuples or simplicial complexes. Another way is to design novel PE/SE for high-order structures from scratch, yet current PE and SE for general k-tuples are limited. In comparison, PE and SE for k-simplicial complexes are better studied and reveal more elegant properties (Zhou et al., 2023a). Following GPS (Rampasek et al., 2022), we choose our PE and SE exactly the same as GPS (including LapPE and RWSE).

Third-Order Variants. Since there are n^3 many 3-tuples, including all tuples $\mathcal{A}_3$ would result in $O(n^6)$ complexity regarding n, which is obviously not practical. Even local neighbor attention $\mathcal{A}_3^{\mathsf{LN}}$ has $O(n^3\bar{D})$ complexity, where $\bar{D}$ is the average node degree. Consequently, we implement $\mathcal{A}_3$ with sampling tuples. Instead of random sampling, we want to take graph structure into account; thus we sample those *connected* 3-tuples. On average there are $O(n\bar{D}^2)$ connected 3-tuples, therefore even dense attention would not exceed $O(n^2\bar{D}^4)$ complexity. However, note that the dense attention again does not provide the connectivity of these sampled 3-tuples, which may have a negative impact on performance. We report the results of sampling connected 3-tuples in Table 12.

Cross Attention. We implement cross attention $\mathcal{A}_{1,2}$ for ablation study. The first-order tensors are updated according to the definition; yet, we also want to update representations of second-order tensors. Inspired by (Sanford et al., 2023) which reconstructs high-order tensors from first-order tensor, we modify $\mathcal{A}_{1,2}$ as follows. Denote the first-order query tensor $\boldsymbol{X} \in \mathbb{R}^n$, second-order key tensor $\boldsymbol{Y} \in \mathbb{R}^{n^2}$, and e_{ij} as the feature of edge connecting nodes i, j. For each layer l the update procedure is as follows,

$$\boldsymbol{X}^{(l+1)} = \mathcal{A}_{1,2}^l(\boldsymbol{X}^l, \boldsymbol{Y}^l) \tag{99}$$

$$e_{ij}^{(l+1)} = \psi_e^l(e_{ij}^l, \boldsymbol{X}_i^{(l+1)}, \boldsymbol{X}_j^{(l+1)}) \tag{100}$$

$$\boldsymbol{Y}_{i,j}^{(l+1)} = \psi_v^l(\boldsymbol{Y}_{i,j}^l, \boldsymbol{X}_i^{(l+1)}, \boldsymbol{X}_j^{(l+1)}, e_{ij}^{(l+1)}) \tag{101}$$

where ψ_e, ψ_v are MLPs, and superscripts $^{(l)}$ refer to the l-th layer. The experimental results of $\mathcal{A}_{1,2}$ are reported in Table 12.

Furthermore, inspired by our sparse attention mechanism for self-attention, we also implement dense (original) and sparse $\mathcal{A}_{1,2}$. In the dense version, all n query tokens compute attention with all n^2 key tokens, resulting in $O(n^3)$ complexity. In the sparse version (denoted as $\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$), every query token only computes attention with those 2-tuples that contain the query node itself, and hence enjoy only $O(n^2)$ complexity. We experimentally find that the sparse $\mathcal{A}_{1,2}$ is not only more efficient in running time, but also usually demonstrates better performance than the dense version. This observation again verifies our motivation: our sparse attention not only reduces computation complexity, but also introduces structure information which improves model performance.

Table 6: Overview of the graph learning datasets used in the paper

Dataset	#Graphs	Avg. # nodes	Avg. # edges	Directed	Prediction level	Prediction task	Metric
Edge detection	12,000	23.2	24.9	No	edge	binary classif.	Accuracy
CSL	150	41	82	No	graph	10-way classif.	Accuracy
Substructure counting	5000	18.8	31.3	No	graph	regression	Mean Abs. Error
WebKB-Cornell	1	183	298	Yes	node	10-way classif.	Accuray
WebKB-Texas	1	183	325	Yes	node	10-way classif.	Accuray
WebKB-Wisconsin	1	251	515	Yes	node	10-way classif.	Accuray
ZINC	12,000	23.2	24.9	No	graph	regression	Mean Abs. Error
Alchemy	202,579	10.0	10.4	No	graph	regression	Mean Abs. Error
ogbg-molhiv	41,127	25.5	27.5	No	graph	binary classif.	AUROC
Peptides-func	15,535	150.9	307.3	No	graph	10-task classif.	Avg. Precision
Peptides-struct	15,535	150.9	307.3	No	graph	11-task regression	Mean Abs. Error

E.2 Dataset Description and Experiments Overview

We provide dataset statistics in Table 6. Our experiments include both synthetic and real-world datasets.

For synthetic datasets, we have already provided the results of edge detection (derived from ZINC (Dwivedi et al., 2020), see (Muller et al., 2023) for more details) and CSL in the main text. We provide an additional substructure counting task in Appendix E.3, since counting substructure is another important capability of graph learning models in addition to expressive power.

For real-world datasets, we conduct extensive experiments on various datasets and report the results in Appendix E.4. We have already reported comprehensive results of almost all our models on ZINC (Dwivedi et al., 2020) in our main text. Additional results include node classification tasks in WebKB (Ghani, 2001), graph-level regression task on Alchemy (Chen et al., 2019), graph-level classification task on OGBG-molhiv (Hu et al., 2020) and two datasets from Long Range Graph Benchmark (LRGB) (Dwivedi et al., 2022).

As emphasized in our main text, since our main goal is to verify the scalability and empirical benefits of our models, we directly adopt experimental settings and hyper-parameters from the survey paper (Muller et al., 2023) for simulation tasks, and from the GraphGPS (Rampasek et al., 2022) paper (a recent SOTA and popular MPNN + graph transformer baseline) for real-world tasks. Even if we do not perform hyperparameter search, our models still reveal highly competitive performances. For computing infrastructure, all our experiments are carried out with NVIDIA GeForce RTX 3090 and 4090. Code available at https://github.com/zhouc20/k-Transformer.

E.3 Simulation Results

The ability of substructure counting is an important perspective in measuring a graph learning model. The ability to count substructures has close connections to theoretical expressive power, but also not completely equivalent. Substructure counting ability may have more detailed and complex results, as well as more practical impacts on real-world tasks. For example, in molecular graphs detecting cycles and rings is crucial in predicting molecule properties.

Substructure Counting. We provide the results of substructure counting in Table 7. The dataset is derived from (Chen et al., 2020) which contains five thousand random regular graphs. There are four target substructures: triangle, tailed triangle, star and chordal-cycle. We measure the performance by MAE, lower MAE corresponding to stronger substructure counting ability. Note that when the model has extremely small MAE (~ 0.01), we regard it capable of counting the target substructure and ignore the difference in absolute MAE value at such a low magnitude. Such a small difference does not necessarily reflect comparison among models - the MAE may depend more on training configurations and hyper-parameters instead of models (and that difference models may vary in their best configurations, so the comparisons are not completely fair).

For baseline models, we choose GCN (Kipf and Welling, 2016), GIN (Xu et al., 2018) PNA (Corso et al., 2020) as

Table 7: Substructure counting performance (test MAE ↓). Highlighted are the first, second and third results.

Model	Triangle	Tailed triangle	Star	Chordal-Cycle
GCN	0.4186	0.3248	0.1798	0.2822
GIN	0.3569	0.2373	0.0224	0.2185
PNA	0.3532	0.2648	0.1278	0.2430
PPGN	0.0089	0.0096	0.0148	0.0090
Transformer	1.0712	0.8929	1.3634	0.9682
$\mathcal{A}_2^{\mathsf{Ngbh}}$	0.3780	0.2764	0.0133	0.2467
$\mathcal{A}_2^{\mathsf{Ngbh}+}$	0.0121	0.0186	0.0146	0.0158
$\mathcal{A}_2^{\mathsf{LN}}$	0.0165	0.0201	0.0458	0.0380
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$	0.0103	0.0133	0.0199	0.1893

the baseline models that are basically equivalent to 1-WL, and PPGN (Maron et al., 2019) that has provable 3-WL equivalent expressive power. We also report the performance of Transformer $\mathcal{A}_1$, i.e. the commonly used plain transformer.

It is known that 1-WL cannot count triangles, tailed triangles and chordal-cycles, but it can count stars (Chen et al., 2020). With their theoretical expressive power upper-bounded by 1-WL, we observe that GCN, GIN and PNA all have relatively huge loss while counting triangles, tailed triangles and chordal-cycles; only GIN with provable 1-WL expressivity can count stars with a small MAE. PPGN can count all substructures well with low MAE (~ 0.01), however at the cost of $O(n^3)$ complexity. The poor performance of the plain transformer $\mathcal{A}_1$ is not surprising: its MAE is almost two magnitudes higher than PPGN, which indicates that $\mathcal{A}_1$ cannot count any substructures. This aligns perfectly with our theory that $\mathcal{A}_1$ is strictly less expressive than 1-WL.

Interestingly, the results of our proposed models also align quite well with our theories, which verify the effectiveness of our sparse high-order transformers.

- We already theoretically proved that $\mathcal{A}_2^{\mathsf{Ngbh}}$ is as powerful as 2-WL (hence 1-WL as well as GIN). The performance of our $\mathcal{A}_2^{\mathsf{Ngbh}}$ model highly resembles that of 1-WL equivalent GIN for all tasks, which is consistent with the theory. $\mathcal{A}_2^{\mathsf{Ngbh}}$ even achieves SOTA on counting stars, which implies the empirical benefit of bringing in global information (recall the definition of 2-neighbor) - although it does not increase theoretical expressive power, the global information helps in algorithmic alignment.

- $\mathcal{A}_2^{\mathsf{Ngbh}+}$ is proved to be strictly more powerful than $\mathcal{A}_2^{\mathsf{Ngbh}}$, which is verified by the experimental results. When counting triangles, tailed triangles and chordal-cycles that $\mathcal{A}_2^{\mathsf{Ngbh}}$ fails, $\mathcal{A}_2^{\mathsf{Ngbh}+}$ all complete perfectly (the same MAE magnitude as PPGN). This verifies that $\mathcal{A}_2^{\mathsf{Ngbh}+}$ is strictly more expressive than 2-WL, otherwise it would not be able to count any of triangles, tailed triangles and chordal-cycles.

- In Appendix B we show that $\mathcal{A}_2^{\mathsf{LN}}$ is strictly less expressive than $\mathcal{A}_2^{\mathsf{Ngbh}+}$, which is attributed to the fact that $\mathcal{A}_2^{\mathsf{LN}}$ has slightly higher MAE compared to $\mathcal{A}_2^{\mathsf{Ngbh}+}$ in all four tasks. However, $\mathcal{A}_2^{\mathsf{LN}}$ is much more efficient than $\mathcal{A}_2^{\mathsf{Ngbh}+}$ as well as PPGN, and the absolute MAE values of $\mathcal{A}_2^{\mathsf{LN}}$ are still at a low magnitude compared with 1-WL baseline models. In summary, $\mathcal{A}_2^{\mathsf{LN}}$ strikes a good balance between expressivity and efficiency.

- The results of the simplicial transformer with simplex neighbor attention mechanism $\mathcal{AS}_{0:1}^{\mathsf{SN}}$ are also consistent with our theory in Appendix C. Leveraging the connectivity of 0-simplices (nodes), 1-simplices (edges), and partially 2-simplices (triangles), $\mathcal{AS}_{0:1}^{\mathsf{SN}}$ performs highly competitive in counting triangles and tailed triangles. However, it fails to count chordal-cycles, which indicates that it is more powerful than 1-WL yet does not achieve full 3-WL expressivity. It has $O((n+m)\bar{D}_{\mathcal{S}})$ complexity where $m = |E|$ is the number of edges, while $\bar{D}_{\mathcal{S}}$ is the average number of extend neighbors (including boundaries, coboundaries, and upper/lower adjacent neighbors); hence $\mathcal{AS}_{0:1}^{\mathsf{SN}}$ is generally even more efficient than $\mathcal{A}_2^{\mathsf{LN}}$ on sparse graphs.

Again, both PPGN and most of our models are completely capable of counting these substructures. However, since PPGN needs internal $O(n^3)$ complexity, our model variants are much more efficient.

Table 8: Results on ogbg-molhiv (Hu et al., 2020). Shown is the mean ± std of 5 runs with different random seeds. Highlighted are the first, second and third results.

Model	AUROC ↑
GCN+virtual node	0.7599 ± 0.0119
GIN+virtual node	0.7707 ± 0.0149
PNA	0.7905 ± 0.0132
DGN (Beaini et al., 2020)	0.7970 ± 0.0097
CIN	0.8094 ± 0.0057
GIN-AK+	0.7961 ± 0.0119
GSN (Bouritsas et al., 2023)	0.8039 ± 0.0090
SAN	0.7785 ± 0.2470
GPS	0.7880 ± 0.0101
$\mathcal{A}_2^{\mathsf{LN+VT}}$ (ours)	0.7901 ± 0.0082
$\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$ (ours)	0.7981 ± 0.0097
$\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ (ours)	0.7981 ± 0.0114

E.4 Additional Results on Real-World Datasets

In addition to the highly competitive performance on ZINC (see the main text), we also provide experimental results on other real-world datasets, including ogbg-molhiv (Hu et al., 2020) and two datasets from Long Range Graph Benchmark (LRGB) (Dwivedi et al., 2022). We directly adopt experimental settings and hyper-parameters from the GraphGPS (Rampasek et al., 2022).

Molecular Property Prediction. For classification task, we choose ogbg-molhiv dataset from the OGB benchmark (Hu et al., 2020), which contains 41k molecules. The task is a binary graph-level classification to predict whether a molecule inhibits HIV virus replication or not. The performance is measured by AUROC.

We report our results in Table 8. It is notable that complex models tend to suffer from overfitting on this dataset, and SOTA models usually contain manually extracted feature, e.g. CIN (Bodnar et al., 2021a) and GSN (Bouritsas et al., 2023) both contain artificially extracted substructures. In comparison, our models completely learn from the graph structures without any additional manually crafted information. In detail, $\mathcal{A}_2^{\mathsf{LN+VT}}$, $\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$ and $\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ all outperform GPS, showing the empirical benefit of high-order models. In particular, the results of $\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$ and $\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ are highly competitive, which implies that global information and local structures may both play an important role in the prediction of graph-level properties.

Table 9: Experiments on two datasets from long-range graph benchmarks (LRGB) (Dwivedi et al., 2022). Shown is the mean ± std of 5 runs with different random seeds. Highlighted are the first, second and third results.

model	Peptides-func (AP ↑)	Peptides-struct (MAE ↓)
GCN	0.5930 ± 0.0023	0.3496 ± 0.0013
GINE	0.5498 ± 0.0079	0.3547 ± 0.0045
GatedGCN	0.5864 ± 0.0077	0.3420 ± 0.0013
Transformer+LapPE	0.6326 ± 0.0126	0.2529 ± 0.0016
SAN+LapPE	0.6384 ± 0.0121	0.2683 ± 0.0043
SAN+RWSE	0.6439 ± 0.0075	0.2545 ± 0.0012
GPS	0.6535 ± 0.0041	0.2500 ± 0.0005
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$ (ours)	0.5876 ± 0.0079	0.2703 ± 0.0015
$\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ (ours)	0.6486 ± 0.0063	0.2524 ± 0.0009
$\mathcal{AS}_{0:1}$-dense+attn.bias (ours)	0.6445 ± 0.0082	0.2486 ± 0.0007

Table 10: Scalability of proposed models on datasets of different scales (Avg. # nodes and Avg. # edges are listed in Table 6). Hyperparmeters are the same as GPS (Rampasek et al., 2022) for all models.

Model	ZINC	ogbg-molhiv	Peptides-func/struct
$\mathcal{A}_2$-dense	✗	✗	✗
$\mathcal{A}_2$-Performer	✓	✗	✗
$\mathcal{A}_2^{\mathsf{Ngbh}}$	✓	✗	✗
$\mathcal{A}_2^{\mathsf{LN}}$	✓	✓	✗
$\mathcal{A}_2^{\mathsf{VT}}$	✓	✓	✓
$\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$	✓	✓	✓
$\mathcal{AS}_{0:1}$-dense	✓	✓	✓
$\mathcal{AS}_{0:1}$-Performer	✓	✓	✓
$\mathcal{AS}_{0:1}^{\mathsf{SN}}$	✓	✓	✓
$\mathcal{AS}_{0:1}^{\mathsf{VS}}$	✓	✓	✓

Long Range Interaction Prediction. Empirically, one of the advantages of graph transformers over message-passing GNNs is that the former are able to capture long-range information via attention, while the latter suffer from restricted reception field. Long Range Graph Benchmark (LRGB) (Dwivedi et al., 2022) is a recently proposed benchmark to evaluate models' capacity in capturing long-range interactions within graphs. We choose two datasets, namely Peptides-func and Peptides-struct from LRGB to evaluate our models. Both datasets consist of atomic graphs of peptides. The task for Peptides-func is a multi-label graph classification into 10 nonexclusive peptide functional classes measure by average precision. The task for Peptides-struct is graph regression of 11 3D-structural properties of the peptides measured by MAE.

As shown in Table 10, since these datasets contain large graphs, $\mathcal{A}_2$ with neighbor attention and local neighbor attention mechanisms cannot scale to them unfortunately. The results of our simplicial transformers are reported in Table 9, while performance of cross-attention and sampling connected 3-tuples are displayed in Table 12 as ablation study. To the best of our knowledge, our simplicial transformers are the first second-order models that scale to LRGB datasets.

According to Table 9, our $\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ and $\mathcal{AS}_{0:1}$-dense+attn.bias both outperform other transformers with PE/SE, including Transformer ($\mathcal{A}_1$) and SAN (Kreuzer et al., 2021). Interestingly, the virtual simplex in $\mathcal{AS}_{0:1}^{\mathsf{SN+VS}}$ leads to a significant performance gain compared with $\mathcal{AS}_{0:1}^{\mathsf{SN}}$, indicating that global information aggregated by the virtual simplex indeed helps model to predict long-range interactions and global properties better. $\mathcal{AS}_{0:1}$-dense+attn.bias internally capture global information due to the dense attention mechanism, and the (extended) Hodge Laplacian as attention bias provides beneficial structural information - it even surpasses GPS on the Peptides-struct dataset.

E.5 Ablation Study and Empirical Analysis

E.5.1 Scalability and Efficiency Analysis

To verify the scalability and efficiency of our proposed models, we summarize the scalability of different models to various datasets in Table 10, and report the number of parameters and the running time of our models on ZINC in Table 11.

As summarized in Table 6, the small molecular graphs in ZINC (Dwivedi et al., 2020) have the smallest average size, while the Peptides-func/struct contain large graphs with hundreds of nodes and edges. Interestingly, $\mathcal{A}_2$ with different attention mechanisms form a step-like scalability in Table 10, which verifies the fact that $\mathcal{A}_2$-dense, $\mathcal{A}_2^{\mathsf{Ngbh}}$, $\mathcal{A}_2^{\mathsf{LN}}$ and $\mathcal{A}_2^{\mathsf{VT}}$ gradually reduce their complexity.

In Table 11, we provide the number of parameters as well as the running time of our models on ZINC. As mentioned in the main text, all these models have similar number of parameters compared with GPS when sharing common hyper-parameters (e.g. number of layers and network width). Regarding running time, all our models have the same magnitude as GPS, which is completely acceptable. Remarkably, our sparse attention

Table 11: # parameters and running time (s/epoch) of our models on ZINC. Other hyper-parameters are the same as in (Rampasek et al., 2022).

Model	# parameters	running time (s/epoch)
GPS (Rampasek et al., 2022)	423,717	21
$\mathcal{A}_2$-Performer	863,269	107
$\mathcal{A}_2^{\mathsf{Ngbh}+}$	322,613	69
$\mathcal{A}_2^{\mathsf{LN}+\mathsf{VT}}$	389,173	37
$\mathcal{A}_{1,2}$-dense	607,253	64
$\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$	601,253	37
$\mathcal{AS}_{0:1}$-dense	342,813	35
$\mathcal{AS}_{0:1}^{\mathsf{SN}+\mathsf{VS}}$	340,069	24

Table 12: Ablation of $\mathcal{A}_3$ with connected 3-tuples and cross-attention $\mathcal{A}_{1,2}$. Shown is mean $\pm$ std of 5 runs with different random seeds.

Model	ZINC (MAE $\downarrow$)	Peptides-func (AP $\uparrow$)	Peptides-struct (MAE $\downarrow$)
$\mathcal{A}_3$(connected)	0.138 ± 0.007	0.6329 ± 0.0117	0.2529 ± 0.0023
$\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$	0.076 ± 0.005	0.6419 ± 0.0108	0.2612 ± 0.0013

mechanisms significantly improve efficiency. Simplicial transformers are generally more efficient than tuple-based transformers. All these observations are consistent with our theoretical analysis.

E.5.2 Cross Attention and Sampling Strategy

In Appendix E.1 we detail the implementation of cross-attention $\mathcal{A}_{1,2}$ and $\mathcal{A}_3$ with sampling connected 3-tuples. The corresponding results are reported in Table 12, where $\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$ refers to the sparse implementation of $\mathcal{A}_{1,2}$ in which each first-order query token computes attention with the 2-tuples containing the query node (i.e., 2-neighbors). We observe that the performances vary from datasets. Specifically, $\mathcal{A}_{1,2}^{\mathsf{Ngbh}}$ achieves competitive 0.076 MAE, in comparison, the dense implementation of $\mathcal{A}_{1,2}$ has 0.091 MAE (not reported in the table). Despite that these methods do not achieve SOTA, they open a chance of leveraging the benefits of higher-order methods in graph learning in the future.

E.5.3 Node Classification and Over-smoothing

It is widely believed that MPNN suffers from problems including over-smoothing and over-squashing, while graph transformers can (partly) address these problems (Muller et al., 2023). However, there is still a lack of thorough and systematical analysis. It is natural to ask whether higher-order graph transformers can address these problems.

Here we primarily and empirically analyze the performance of high-order transformers in heterophilic transductive node classification tasks on heterogeneous graphs. Cornell, Texas, and Wisconsin are three popular datasets from WebKB (Ghani, 2001). We follow the experimental settings and hyperparameters of (Muller et al., 2023), and report the performance of two-layer simplicial transformers in Table 13. We do not report tuple-based transformers since the graphs are directed, thus the concept of tuples is not applicable.

Our simplicial transformer without PE/SE significantly outperforms GPS with PE/SE, verifying the advantages of the global attention mechanism. Our models also have similar performance compared to Transformer with RWSE/LapPE, leveraging the advantage of structure awareness brought by attention bias. An interesting phenomenon is that Transformer alone perform better without GCN - adding GCN as in GPS would even decrease the test accuracy. Muller et al. (2023) anticulated that this is due to the oversmoothing of message-passing GNNs on heterophilic graph. Our results somewhat support this explanation; however, more future work is needed to

Table 13: Node classification performance on three heterophilic transductive datasets. Shown is the mean ± std of 10 runs with different random seeds.

Model	Cornell	Texas	Wisconsin
GCN	53.78 ± 3.07	65.95 ± 3.67	66.67 ± 2.63
GCN+LapPE	56.22 ± 2.65	65.95 ± 3.67	66.47 ± 1.37
GCN+RWSE	53.78 ± 4.09	62.97 ± 3.21	69.41 ± 2.66
GCN+DEG	53.51 ± 2.65	66.76 ± 2.72	67.26 ± 1.53
GPS(GCN+Transformer)+LapPE	66.22 ± 3.87	75.41 ± 1.46	74.71 ± 2.97
GPS(GCN+Transformer)+RWSE	65.14 ± 5.73	73.51 ± 2.65	78.04 ± 2.88
GPS(GCN+Transformer)+DEG	64.05 ± 2.43	73.51 ± 3.59	75.49 ± 4.23
Transformer+LapPE	69.46 ± 1.73	77.84 ± 1.08	76.08 ± 1.92
Transformer+RWSE	70.81 ± 2.02	77.57 ± 1.24	80.20 ± 2.23
Transformer+DEG	71.89 ± 2.48	77.30 ± 1.32	79.80 ± 0.90
Graphormer+DEG	68.38 ± 1.73	76.76 ± 1.79	77.06 ± 1.97
Graphormer+attn.bias	68.38 ± 1.73	76.22 ± 2.36	77.65 ± 2.00
$\mathcal{AS}_{0:1}$-dense+attn.bias	70.27 ± 2.96	76.84 ± 1.66	77.45 ± 0.98

give a comprehensive answer to the question. Moreover, it is also worth exploring the empirical advantages of (high-order) transformers over MPNN and other non-transformer models.