Efficient Higher Order Self Attention via Tensor Operations with Topology Enhanced Graph Neural Network for Molecular Graph Classification

by

Choi Wan Ioi DC011407

MATH4002

Research in Mathematics

2023/2024

BSc in Mathematics (Mathematics and Applications)

Department of Mathematics

Faculty of Science and Technology

University of Macau

Name of Supervisor: Kou Kit Ian

Faculty/Department: Faculty of Science and Technology / Department of Mathematics

Name of Co-Supervisor (if any): ______________________________

Faculty/Department: ______________________________

Approved by ____

Efficient Higher Order Self Attention via Tensor Operations with Topology Enhanced Graph Neural Network for Molecular Graph Classification

by Choi Wan Ioi

Thesis Supervisor: Professor Kou Kit Ian

Abstract

We modify a recently proposed higher order generalization of self attention mechanism and reduce its time complexity using tensor operations. Then the modified higher order self attention is applied to a graph neural network with a novel structural encoding based on cell complex from algebraic topology. Experiments on molecular graph benchmarks show that our modified higher order self attention is more efficient than the original higher order self attention, and our proposed structural encoding improves the performance of the graph neural network.

Keywords self attention, tensor decomposition, molecular property prediction, graph neural network, cell complex

Contents

Section 1

Introduction

In recent years, there has been a remarkable surge in the field of natural language processing and computer vision, thanks to the advancements in deep learning. One significant breakthrough that has revolutionized these domains is the introduction of the self-attention mechanism [1]. Self-attention has emerged as a powerful tool for modeling dependencies and capturing contextual information within a sequence of data, be it text or images.

The key idea behind self-attention is to calculate attention weights by inner product between every pair of positions in the input sequence. These attention weights represent the importance or relevance of each position with respect to others. The attention weights are then used to compute weighted sums of the values associated with each position, resulting in a context vector that encapsulates the relevant information from the entire sequence.

Intuitively, self attention captures pairwise interaction between positions in a sequence, but not higher order interaction. Recent work [2] has shown that self attention is unable to solve a simple task about learning the correlation between triples of words. To address this issue, the authors proposed a higher order self attention, but it was not implemented due to the expensive computational cost.

Graph neural networks have emerged as an effective approach for tasks over graph-structured data, and structural encoding is crucial to the effectiveness of graph neural network. Motivated by the success of cell complex neural networks which are generalization to graph neural networks that can model higher order interactions, it is interesting to explore whether incorporating structural encoding based on cell complex can enhance the performance of graph neural networks.

Contribution We modified the higher order self attention in [2] and implement it using tensor operations to reduce its time complexity from $O(N^2d^2)$ to $O(Nd^2)$. We also proposed a novel structural encoding for graph neural network which is an extension to the degree encoding. Experiments on various molecular graph datasets show that our modified higher order self attention is more efficient than the original formulation of the higher order self attention, and our proposed structural encoding can effectively enhance the performance of the graph neural network.

Section 2

Preliminary

2.1 Tensor Operations and Decompositions

In this subection, we will introduce some basic tensor operations and their properties, such as mode-n matricization, mode-n product and tensor contraction. We will also introduce the CP decomposition of a tensor and its properties.

We use script letters (e.g. $\mathcal{A}, \mathcal{B}$) to represent tensors and use italic letters (e.g. A, B) to represent matrices.

Definition 1 (mode-n matricization [3]) The mode-n matricization of a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ is a matrix $\mathcal{T}_{(n)} \in \mathbb{R}^{I_n \times I_1 I_2 \cdots I_{n-1} I_{n+1} \cdots I_N}$.

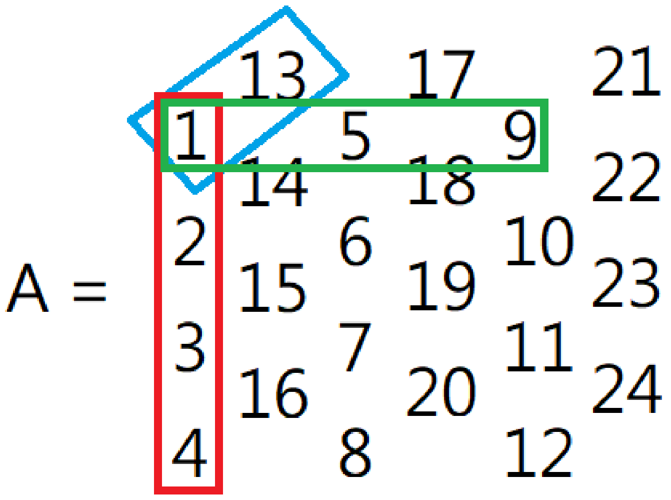


Figure 1: A tensor of size 4x3x2

It is easier to understand the concept of mode-n matricization using an example. Let $\mathcal{A} \in \mathbb{R}^{4\times3\times2}$ be the tensor as shown in figure 1. Then

$$\mathcal{A}_{(1)} = \begin{bmatrix} 1 & 5 & 9 & 13 & 17 & 21 \\ 2 & 6 & 10 & 14 & 18 & 22 \\ 3 & 7 & 11 & 15 & 19 & 23 \\ 4 & 8 & 12 & 16 & 20 & 24 \end{bmatrix}$$

$$\mathcal{A}_{(2)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 13 & 14 & 15 & 16 \\ 5 & 6 & 7 & 8 & 17 & 18 & 19 & 20 \\ 9 & 10 & 11 & 12 & 21 & 22 & 23 & 24 \end{bmatrix}$$

$$\mathcal{A}_{(3)} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \end{bmatrix}$$

The mode-n product between a tensor and a matrix can be defined using the mode-n matricization.

Definition 2 (mode-n product [3]) The mode-n product between a tensor $\mathcal{T} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ and a matrix $A \in \mathbb{R}^{J\times I_n}$ results in a tensor $\mathcal{T} \times_n A \in \mathbb{R}^{I_1\times\cdots\times I_{n-1}\times J\times I_{n+1}\times\cdots\times I_N}$ with

$$(\mathcal{T} \times_n A)_{(n)} = A\mathcal{T}_{(n)}$$

Some useful properties of the mode-n product are included in the following lemma.

Lemma 1 (Properties of mode-n product [3]) Let $\mathcal{T} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ and $A_n \in \mathbb{R}^{J_n\times I_n}$.

1. $\mathcal{T} \times_m A_m \times_n A_n = \mathcal{T} \times_n A_n \times_m A_m$ for $m \neq n$
2. For $B_n \in \mathbb{R}^{K_n\times J_n}$, $\mathcal{T} \times_n A_n \times_n B_n = \mathcal{T} \times_n (B_n A_n)$

The CP decomposition of a tensor can be defined using the mode-n product.

Definition 3 (CP decomposition [4]) The CP decomposition decompose a tensor $\mathcal{T} \in \mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ into a superdiagonal tensor $\mathcal{I}_R \in \mathbb{R}^{R\times\cdots\times R}$ with diagonal $1, \dots, 1$ multiplied by a factor matrix $A_n \in \mathbb{R}^{I_n\times R}$ on each mode as

$$\mathcal{T} = \mathcal{I}_R \times_1 A_1 \times_2 A_2 \cdots \times_n A_N$$

where $R \in \mathbb{N}$ is called the CP rank of $\mathcal{T}$.

The relation between the mode-n matricization of a tensor $\mathcal{T}$ and its CP decomposition is given in the following lemma.

Lemma 2 ([3]) $\mathcal{T} = \mathcal{I}_R \times_1 A_1 \times_2 A_2 \cdots \times_N A_n \Leftrightarrow \mathcal{T}_{(n)} = A_n(A_N \odot \cdots \odot A_{n+1} \odot A_{n-1} \odot \cdots \odot A_1)^T$, where $\odot$ is the column-wise Kronecker product.

We also introduce the definition of tensor contraction.

Definition 4 (Tensor contraction) Given $\mathcal{A} \in \mathbb{R}^{I_1 \times K_1 \times K_2}$ and $\mathcal{B} \in \mathbb{R}^{J_1 \times K_1 \times K_2}$. A tensor contraction over 2 modes results in a tensor $\mathcal{A} \cdot \mathcal{B} \in \mathbb{R}^{I_1 \times J_1}$ with

$$(\mathcal{A} \cdot \mathcal{B})_{i_1 j_1} = \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} \mathcal{A}_{i_1 k_1 k_2} \mathcal{B}_{j_1 k_1 k_2}$$

2.2 Cell Complex

In this subsection, we will introduce the definition of cell complex, a fundamental building block of algebraic topology.

Definition 5 (Regular cell complex [5]) A regular cell complex is a topological space X together with a partition $\{X_\sigma\}_{\sigma \in P_X}$ of subspaces X_σ of X such that:

1. For each $x \in X$, every sufficiently small neighborhood of x intersects finitely many X_σ.
2. For all $\tau, \sigma \in P_X$, we have that $X_\tau \cap \overline{X_\sigma} \neq \emptyset$ if and only if $X_\tau \subseteq \overline{X_\sigma}$.
3. Every X_σ is homeomorphic to $\mathbb{R}^{n_\sigma}$ for some n_σ.
4. For every $\sigma \in P_X$, there is a homeomorphism ϕ of a closed ball in $\mathbb{R}^{n_\sigma}$ to $\overline{X_\sigma}$ such that the restriction of ϕ to the interior of the ball is a homeomorphism onto X_σ.

In addition, we define $\tau \leq \sigma \Leftrightarrow X_\tau \subseteq \overline{X_\sigma}$ and $\tau < \sigma \Leftrightarrow X_\tau \subset \overline{X_\sigma}$.

A k-cell is a cell $\sigma \in P_X$ with $n_\sigma = k$. For example, a 0-cell is a point, a 1-cell is a line segment without its endpoint, and a 2-cell is the interior of a polygon.

A cell complex X is of dimension k if $n_\sigma \leq k$ for all $\sigma \in P_X$ and there is at least one $\tau \in P_X$ such that $n_\tau = k$. An example of a cell complex of dimension 2 is shown in Figure 2.

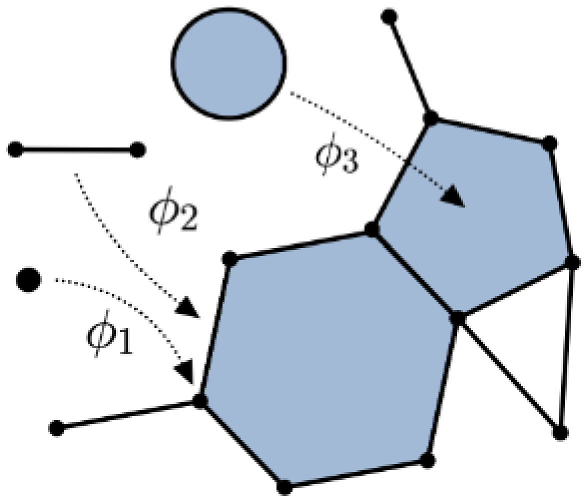


Figure 2 ([5]): A cell complex of dimension 2

In practice, we only consider cell complexes of dimension $k \leq 2$. A cell complex of dimension 2 can be represented by a 3-tuple (V, E, P), where V, E, P are the sets of 0-cells (vertices), 1-cells (edges) and 2-cells (polygons) respectively. Also, a graph $G = (V, E)$ can be seen as a cell complex of dimension 1, where the set of 0-cells is the set of vertices V and the set of 1-cells is the set of edges E.

Intuitively, a cell complex is formed by gluing cells together in a way that every $(k-1)$-cell is either on the boundary of some k-cells, or has no intersection with the boundary of any k-cells. Therefore, the structure of a cell complex can be described by a boundary relation. The boundary relation describes which cells are on the boundary of other cells.

Definition 6 (Boundary relation [5]) Given two cells $\sigma, \tau \in P_X$, we have the boundary relation $\sigma \prec \tau$ if $\sigma < \tau$ and there is no δ such that $\sigma < \delta < \tau$.

We can use the boundary relation to define four types of adjacencies present in cell complexes.

Definition 7 (Cell complex adjacencies [5]) For a cell complex X and a cell $\sigma \in P_X$, we define:

1. The boundary adjacent cells $B(\sigma) = \{\tau \mid \tau \prec \sigma\}$. These are the lower-dimensional cells on the boundary of σ. For example, the boundary adjacent cells of an edge are its vertices, and the boundary adjacent cells of a polygon are its edges.

2. The co-boundary adjacent cells $C(\sigma) = \{\tau \mid \sigma \prec \tau\}$. These are the higher-dimensional cells with σ on its boundary. For example, the co-boundary adjacent cells of a vertex are the edges having the vertex as an endpoint, and the co-boundary adjacent cells of an edge are the polygons having the edge as one of its sides.
3. The lower adjacent cells $N_l(\sigma) = \{\tau \mid \exists\delta \textit{ such that } \delta \prec \sigma \textit{ and } \delta \prec \tau\} = \{\tau \mid B(\sigma) \cap B(\tau) \neq \emptyset\}$. These are the cells of the same dimension as σ with a lower dimensional cells on the boundary of σ on their boundary. For example, the lower adjacent cells of an edge e_i are edges that shares a common endpoint with e_i, and the lower adjacent cells of a polygon p_i are polygons that shares a common side with p_i.
4. The upper adjacent cells $N_u(\sigma) = \{\tau \mid \exists\delta \textit{ such that } \sigma \prec \delta \textit{ and } \tau \prec \delta\} = \{\tau \mid C(\sigma) \cap C(\tau) \neq \emptyset\}$. These are the cells of the same dimension as σ on the boundary of a higher-dimensional cells with σ on its boundary. For example, the upper adjacent cells of a vertex v_i are vertices that are connected to v_i by an edge, and the upper adjacent cells of an edge e_i are edges that are the sides of polygons with e_i as one of its sides.

Corollary 1 Given a graph $G = (V, E)$ and a vertex v_i, the number of co-boundary adjacent cells $|C(v_i)|$ and the number of upper adjacent cells $|N_u(v_i)|$ of v_i is equal to $\deg v_i$. Also the number of boundary adjacent cells $|B(v_i)|$ and the number of lower adjacent cells $|N_l(v_i)|$ of a vertex v_i is equal to 0.

Given a graph $G = (V, E)$, one can construct an associated cell complex of dimension 2 by attaching 2-cells to all chordless cycles in the graph as shown in Figure 3.

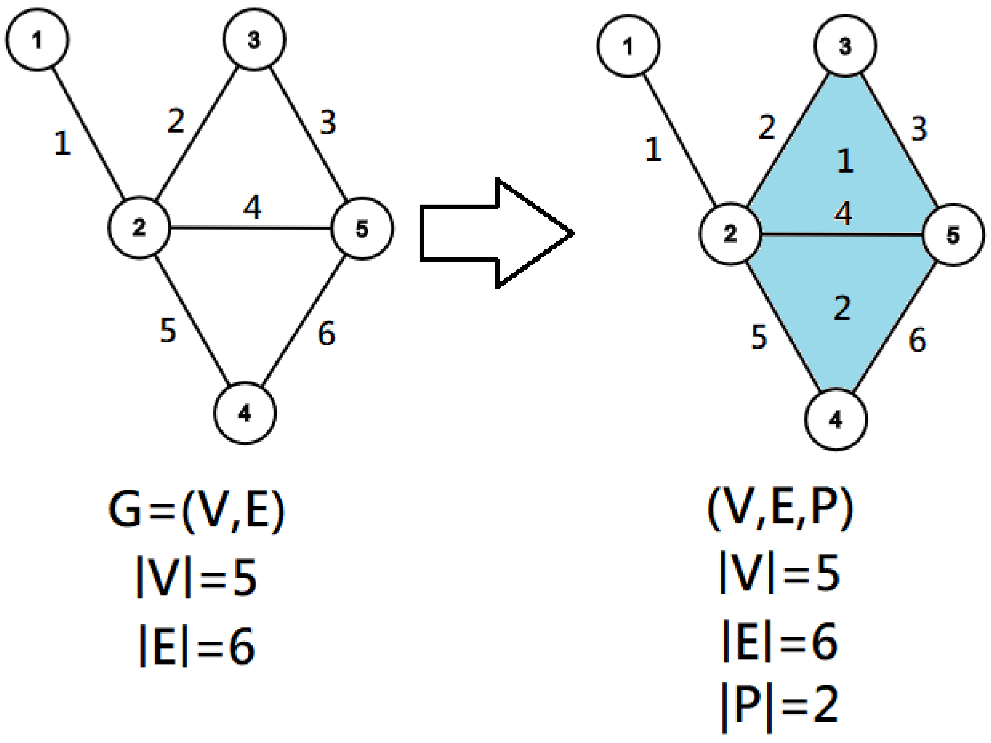


Figure 3: A graph and its associated cell complex of dimension 2

Definition 8 (Chordless cycle) A chordless cycle in a graph is a cycle such that no two vertices of the cycle are connected by an edge that does not itself belong to the cycle.

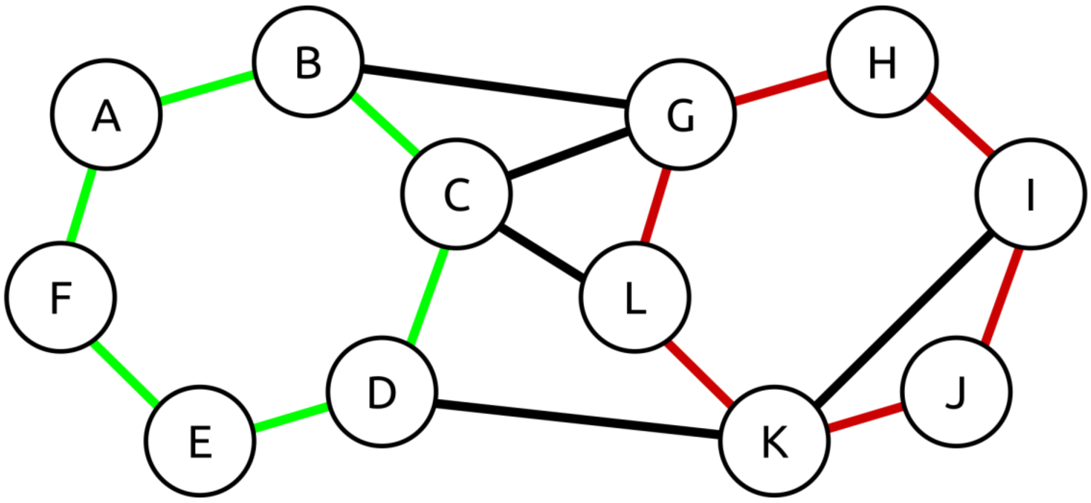


Figure 4: A chordless cycle (green) and a cycle that is not chordless (red)

Section 3

Self attention and its generalization

Self attention [1], since its proposal, have been widely used in the field of natural language processing and computer vision. Based on self attention, famous large language model and generative model, such as GPT-4 and Sora, have been developed.

In this section, we will introduce the vanilla self attention and a recently proposed higher order generalization of self attention.

Definition 9 (self attention [1]) Given N input vectors with dimension d, written in matrix form $X \in \mathbb{R}^{N\times d}$, the self attention $f: \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ computes

$$f(X) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where $Q = XW_Q$, $K = XW_K$ and $V = XW_V$ are three different learnable linear transformation of X.

This can be rewritten as

$$f(X)_i = \sum_{j=1}^{N} A_{ij}V_j = \left(\sum_{j=1}^{N} A_{ij}X_j\right)W_V$$

where

$$A_{ij} = softmax\left(\frac{QK^T}{\sqrt{d}}\right)_{ij} = \frac{\exp\left(X_i W_Q W_K^T X_j^T/\sqrt{d}\right)}{\sum_{j=1}^{N}\exp\left(X_i W_Q W_K^T X_j^T/\sqrt{d}\right)}$$

Essentially, self attention updates each input vector by computing a weighted sum of all input vectors followed by a linear transformation, where the weights depend on every pair of input vector.

Intuitively, self attention can model pairwise interaction in the input vectors since the effect of a specific input vector on the output is affected by the values of other input vectors. However, it is unclear whether self attention can model triplewise or higher order interaction in the input vectors.

Recently, [2] has shown that self attention cannot capture triplewise interaction. They defined a task related to learning correlation between triples of words, and show that it cannot be solved using self attention. To deal with this limitation, they proposed a higher order generalization of self attention using column-wise Kronecker product.

Definition 10 (s-order self attention [2]) For order $s \geq 2$, input matrix $X \in \mathbb{R}^{N\times d}$ and matrices $Q, K_1, K_2, V_1, V_2 \in \mathbb{R}^{d\times d}$, s-order self attention $f\colon \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ computes

$$f(X) = softmax\left(\underbrace{XQ}_{\mathbb{R}^{N\times d}} \underbrace{\big((XK_1) \odot \cdots \odot (XK_{s-1})\big)^T}_{\mathbb{R}^{d\times N^{s-1}}}\right)\underbrace{\big((XV_1) \odot \cdots \odot (XV_{s-1})\big)}_{\mathbb{R}^{N^{s-1}\times d}}$$

When $s = 3$, we obtain the third order self attention.

$$f(X) = softmax\left(\underbrace{XQ}_{\mathbb{R}^{N\times d}} \underbrace{\big((XK_1) \odot (XK_2)\big)^T}_{\mathbb{R}^{d\times N^2}}\right)\underbrace{\big((XV_1) \odot (XV_2)\big)}_{\mathbb{R}^{N^2\times d}}$$

The author in [2] did not implement the third order attention due to its expensive computational cost when computing $\underbrace{XQ}_{\mathbb{R}^{N\times d}} \underbrace{\big((XK_1) \odot (XK_2)\big)^T}_{\mathbb{R}^{d\times N^2}}$ which requires $O(N^3 d)$ time if the classic matrix multiplication algorithm is used.

Section 4

Efficient third order self attention

In this section, we will simplify and rewrite the third order attention using tensor operations.

We first simplify the third order self attention by omitting the exponential function in the softmax operator. The simplified third order attention computes

$$f(X) = D^{-1}\left(\underbrace{XQ}_{\mathbb{R}^{N\times d}}\underbrace{\left((XK_1)\odot(XK_2)\right)^T}_{\mathbb{R}^{d\times N^2}}\right)\underbrace{\left((XV_1)\odot(XV_2)\right)}_{\mathbb{R}^{N^2\times d}}$$

where $D = diag\left(\underbrace{XQ}_{\mathbb{R}^{N\times d}}\underbrace{\left((XK_1)\odot(XK_2)\right)^T}_{\mathbb{R}^{d\times N^2}}\mathbf{1}_{N^2\times 1}\right) \in \mathbb{R}^{N\times N}$ can be computed in $O(N^2 d)$. However the product $\underbrace{\left((XK_1)\odot(XK_2)\right)^T}_{\mathbb{R}^{d\times N^2}}\underbrace{\left((XV_1)\odot(XV_2)\right)}_{\mathbb{R}^{N^2\times d}}$ still requires $O(N^2 d^2)$ time. Therefore, we rewrite the simplified third order self attention using tensor operations.

Theorem 1 Let $\mathcal{A} = \mathcal{I}_d \times_1 Q \times_2 K_2 \times_3 K_1$ and $\mathcal{B} = \mathcal{I}_d \times_2 V_2 \times_3 V_1$. Then $(\mathcal{A} \times_1 X \times_2 X \times_3 X)_{(1)} = XQ\left((XK_1)\odot(XK_2)\right)^T$ and $(\mathcal{B} \times_2 X \times_3 X)_{(1)} = \left((XV_1)\odot X(V_2)\right)^T$.

Proof: $\mathcal{A} \times_1 X \times_2 X \times_3 X = \mathcal{I}_d \times_1 Q \times_2 K_2 \times_3 K_1 \times_1 X \times_2 X \times_3 X$

$= \mathcal{I}_d \times_1 Q \times_1 X \times_2 K_2 \times_2 X \times_3 K_1 \times_3 X$ (Lemma 1 property 1)

$= \mathcal{I}_d \times_1 XQ \times_2 XK_2 \times_3 XK_1$ (Lemma 1 property 2)

Therefore $(\mathcal{A} \times_1 X \times_2 X \times_3 X)_{(1)} = (\mathcal{I}_d \times_1 XQ \times_2 XK_2 \times_3 XK_1)_{(1)}$

$= XQ\left((XK_1) \odot (XK_2)\right)^T$ (Lemma 2)

The proof for $(\mathcal{B} \times_2 X \times_3 X)_{(1)} = \left((XV_1) \odot X(V_2)\right)^T$ is similar.

Corollary 2 Let $\mathcal{A} = \mathcal{I}_d \times_1 Q \times_2 K_2 \times_3 K_1$ and $\mathcal{B} = \mathcal{I}_d \times_2 V_2 \times_3 V_1$. Then $\left((\mathcal{A} \times_1 X \times_2 X \times_3 X \times_1 D^{-1}) \cdot (\mathcal{B} \times_2 X \times_3 X)\right)_{(1)} = D^{-1}XQ\left((XK_1) \odot (XK_2)\right)^T\left((XV_1) \odot X(V_2)\right)$

The advantage of this formulation of the simplified third order attention is that the CP decomposition of the mode-n product $\mathcal{A} \times_n X$ and the tensor contraction $\mathcal{A} \cdot \mathcal{B}$ can be computed from the CP decomposition of $\mathcal{A}$ and $\mathcal{B}$ efficiently [6]. For example, the CP decomposition of the mode-n product $\mathcal{A} \times_1 X$ can be computed in $O(Nd^2)$ time using Lemma 1 property 2 given the CP decomposition of $\mathcal{A} = \mathcal{I}_d \times_1 Q \times_2 K_2 \times_3 K_1$. Also, $(\mathcal{A} \times_1 X \times_2 X \times_3 X \times_1 D^{-1}) \cdot (\mathcal{B} \times_2 X \times_3 X)$ and D (which can be thought as computing the tensor contraction $(\mathcal{A} \times_1 X \times_2 X \times_3 X) \cdot \mathbf{1}_{1\times N\times N}$) can be computed in $O(Nd^2)$ time using similar technique.

We introduce our proposed efficient third order self attention layer.

Definition 11 (efficient third order self attention) Given $X \in \mathbb{R}^{n\times d}$, the efficient third order self attention $f: \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ computes

$$f(X) = (\mathcal{A} \times_1 X \times_2 X \times_3 X \times_1 D^{-1}) \cdot (\mathcal{B} \times_2 X \times_3 X)$$

where $\mathcal{A}$ and $\mathcal{B}$ are tensors with CP rank d and learnable factor matrices. The elements in the factor matrices of $\mathcal{A}$ are initialized from $N(0,1/d^4)$ and the elements in the factor matrices of $\mathcal{B}$ are initialized from $N(0,1/d^3)$.

We fix the factor matrices of $\mathcal{A}$ and $\mathcal{B}$ to have CP rank d to ensure the efficient third order self attention have approximately the same number of learnable parameters as the original third order self attention. The elements in the factor matrices of $\mathcal{A}$ and $\mathcal{B}$ are initialized in a way that the elements of $(\mathcal{A} \times_1 X \times_2 X \times_3 X \times_1 D^{-1})$ and the elements of $(\mathcal{B} \times_2 X \times_3 X)$ will have approximately unit variance when the elements of X have unit variance.

Section 5

Graph Neural Network

In this section, we will describe how to apply the efficient third order attention in a Graph Neural Network (GNN) for graph classification.

Given a graph $G = (V, E)$ with node attributes $X \in \mathbb{R}^{|V| \times d}$, we aim to predict the class which the graph belongs to. This usually involves the following three steps.

Step 1: Use a GNN to update the node representation, i.e., $\hat{X} = GNN(X, G) \in \mathbb{R}^{|V| \times d}$

Step 2: Use a permutation-invariant readout function (e.g. mean, sum, max) to obtain the graph representation, i.e., $\hat{Y} = readout(\hat{X}) \in \mathbb{R}^{d}$

Step 3: The graph representation $\hat{Y}$ is further transformed depending on the task. For a graph classification task, $\hat{Y}$ is mapped to a one-hot encoded vector.

GNNs can be divided into 3 caterogies: Message Passing Neural Network (MPNN), Graph Transformer (GT) and Spectral Graph Neural Network (SGNN).

MPNNs update node representations by aggregating information from neighboring nodes. The node representation of node v_i is updated by

$$\hat{X}_i = f\left(X_i, aggregate_{v_j \in N(v_i)}\left(g(X_i, X_j)\right)\right)$$

where f and g are learnable functions, $aggregate$ is a permutation-invariant function and $N(v_i)$ is the neighborhood of node v_i. By stacking k layers of MPNN, the node representation $\hat{X}_i$ can capture information from the k-hop neighborhood of node v_i.

GTs update node representations by combining information from all nodes in the graph using self attention mechanism.

$$\hat{X}_i = SelfAttention(X)_i$$

The updated node representation by GTs of node v_i depends on the node representation of all nodes in the graph, unlike the updated node representation by MPNNs which only depends on the node representation of the neighborhood of node i. Therefore GTs can capture longer range interaction than MPNNs but with a higher computational cost.

Different from MPNNs and GTs, SGNNs update the node representation by utilizing the eigendecomposition of the symmetrically normalized graph laplacian.

Definition 12 (symmetrically normalized graph laplacian) For a graph $G = (V, E)$ with N nodes, the symmetrically normalized graph laplacian $L \in \mathbb{R}^{N\times N}$ is defined as

$$L_{i,j} = \begin{cases} 1 & if\ i = j\ and\ \deg v_i \neq 0 \\ -\dfrac{1}{\sqrt{\deg v_i \deg v_j}} & if\ i \neq j\ and\ v_i\ is\ adjacent\ to\ v_j \\ 0 & otherwise \end{cases}$$

Since the symmetrically normalized graph laplacian is symmetric, it admits an eigendecomposition $L = U\Lambda U^T$ where $UU^T = U^TU = I_N$ and $\Lambda = diag(\lambda) = diag(\lambda_1, \dots, \lambda_n)$.

SGNNs update the node representation in three steps:

Step 1: $\bar{X} = U^TX$ (Graph Fourier Transform)

Step 2: $\tilde{X} = f(\Lambda)\bar{X}$ (Filtering)

Step 3: $\hat{X} = U\bar{X}$ (Inverse Graph Fourier Transform)

Therefore the updated node representation can be written as $\hat{X} = Uf(\Lambda)U^TX$.

In SGNNs, f is usually a polynomial acting elementwisely on the diagonal of Λ, such as Chebyshev polynomial [7], Bernstein polynomial [8] and Jacobi polynomial [9]. Using the property $Up(\Lambda)U^T = p(U\Lambda U^T) = p(L)$ where p is a polynomial, polynomial SGNN can avoid computing the expensive eigenvalue decomposition. Also, a k-th order polynomial SGNN updates the node representation of a node v_i using information in its k-hop neighborhood, due to the fact that $L_{i,j}^k \neq 0$ when v_j is in the k-hop neighborhood of v_i.

Recently, [10] proposed Specformer by replacing f by self attention mechanism. The eigenvalues $\lambda \in \mathbb{R}^N$ is mapped to vectors $\hat{\lambda} \in \mathbb{R}^{N \times d}$, and $\hat{\lambda}$ is updated by self attention instead of a polynomial. They showed that Specformer can outperform polynomial SGNN due to the fact that self attention can capture interaction between eigenvalues while the elementwise polynomial cannot.

Section 6

Cell Complex Neural Network

In this section, we will introduce cell complex neural network, which is generalization of graph neural network to cell complex.

Cell complex neural network is motivated by the fact that graph can only model pairwise relation between nodes, therefore unable to model long-range and group interactions consistently. On the other hand, cell complex can model higher-dimensional structures and higher-order interaction, which is crucial for applications such as social network and molecular chemistry, since social interaction can involve more that two people and electrostatic interaction can involve more that two atoms.

[5] proposed CIN which is a MPNN on cell complex. CIN computes a cell representation (including node representation, edge representation and polygon representation) for every cell in the associated cell complex of dimension 2 constructed from a molecular graph. Then the cell representation is updated by aggregating information from the cell's boundary neighborhood, co-boundary neighborhood, lower neighborhood and upper neighborhood simultaneously. Finally, readout functions are applied to the node representations, edge representations and polygon representations respectively to obtain three different graph representations, which are further combined to obtain the final graph representation.

[11] proposed CAN which is also a MPNN on cell complex. Different from CIN, CAN first combine the node representations with the edge representations. Then only the edge representations are updated by combining information from the edges in the lower neighborhood and upper neighborhood of an edge. Finally, a readout function is applied to the edge representations to obtain the graph representation.

Section 7

Topological Structural Encoding

In this section, we will introduce structural encoding for graph transformers (GTs) and our proposed topological structural encoding.

Recall that GTs update node representations by combining information from all nodes in the graph (see section 5). Therefore, GTs can be seen as MPNNs on a fully connected graph. This allows GTs to capture long range interactions, but at the cost of losing local information. This is because on a fully connected graph, the shortest path distance between every two nodes becomes 1, the degree of every node becomes $|V| - 1$, and the k-hop neighborhood of every node becomes the same. Without extra information, GTs will not be able to recover important local information that are crucial for prediction.

Structural encoding aims to compensate for the missing local information in GTs by concatenating the input node attributes with a vector that contains local structure information. To accomplish this, the structural encoding of a node should depend on some k-hop neighborhood of the node. Also, the structural encoding of two node should be close when the k-hop neighborhood of the two node are similar.

One of the simpliest structural encoding is the degree encoding. Degree encoding computes the degree of a node $\deg v_i$ and concatenate it with the corresponding node attribute X_i, and can provide information about the 1-hop neighborhood of a node. Another structural encoding is the random-walk encoding [12]. Random-walk encoding computes the k-step random-walk matrix and uses its diagonal as structural encoding.

Definition 13 (k-step random-walk matrix) The k-step random-walk matrix is defined as

$$\hat{A}^k = (D^{-1}A)^k$$

where A is the adjacency matrix and D is the degree matrix.

The element $\hat{A}_{ii}^k$ on the diagonal of the k-step random-walk matrix represents the k-step return probability of a random walk starting from node v_i, which depends on the structure of the k-hop neighborhood of v_i. Therefore $\hat{A}_{ii}^k$ can be used as structural encoding for v_i. In practice, $\hat{A}^1, \hat{A}^2, \cdots, \hat{A}^k$ are all computed, and the diagonal elements $\hat{A}_{ii}^1, \hat{A}_{ii}^2, \cdots, \hat{A}_{ii}^k$ are concatenated with the input node attribute X_i.

Motivated by the success of cell complex neural network and the necessity of structural encoding, we proposed a novel structural encoding based on cell complex.

Definition 14 (Topological structural encoding) Given a cell complex of dimension 2 (V, E, P), the topological structural encoding of an edge e_i is defined as

$$TSE(e_i) = \begin{bmatrix} |C(e_i)| \\ |N_l(e_i)| \\ |N_u(e_i)| \end{bmatrix}$$

Different from degree encoding and random-walk encoding, our proposed TSE is concatenated with the input edge attributes instead of node attributes. It can be seen that TSE is equivalent to degree encoding if the edge e_i is changed to node v_i in the definition of TSE by corollary 1 in section 2. Therefore, our proposed TSE can be seen as a natural generalization of degree encoding to edges.

Section 8

Experiments

In this section, we replace the self attention of Specformer introduced in section 5 by our efficient third order self attention (ETOSA), and test our proposed TSE on 8 small datasets from ADMET benchmark group [13]. ADMET benchmark group consists of 22 datasets for prediction of absorption, distribution, metabolism, excretion and toxicity (ADMET) properites of drugs. Results are shown in Table 1 and Table 2.

We also compared our efficient third order self attention with the original third order self attention in terms of computation time and memory usage on the DILI dataset. Results are shown in Table 3.

Details for datasets and experimental setup can be found in Appendix.

	Bioavailability	CYP2C9 Substrate	CYP2D6 Substrate	CYP3A4 Substrate
Specformer	0.620 ± 0.038	0.402 ± 0.022	0.673 ± 0.029	0.611 ± 0.023
Specformer + TSE	0.647 ± 0.029	0.426 ± 0.023	0.696 ± 0.012	0.604 ± 0.023
Specformer + ETOSA	0.617 ± 0.015	0.423 ± 0.020	0.662 ± 0.028	0.593 ± 0.021
Specformer + TSE + ETOSA	0.636 ± 0.027	0.412 ± 0.028	0.653 ± 0.033	0.592 ± 0.036

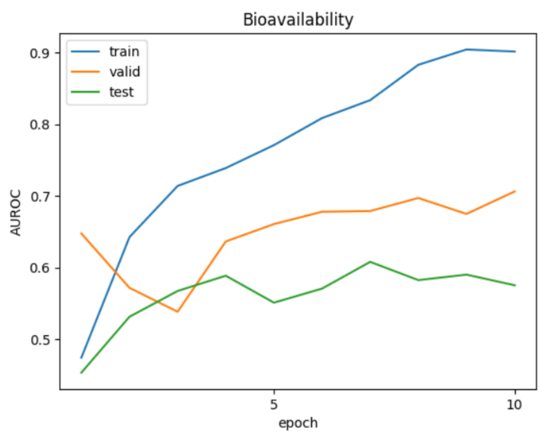
Table 1: Results on ADMET benchmark group. Shown is the mean ± s.d. for 5 runs.

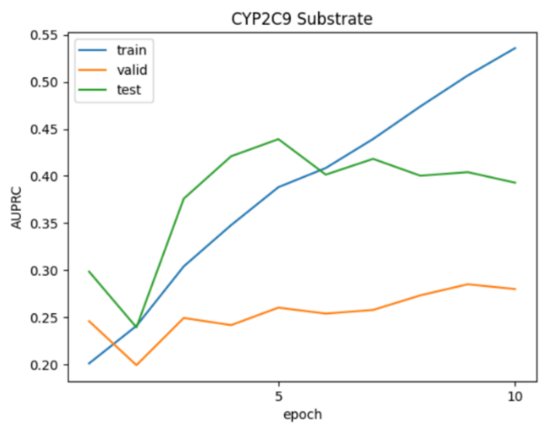
	DILI	hERG	HIA	Pgp
Specformer	0.912 ± 0.003	0.813 ± 0.043	0.987 ± 0.005	0.889 ± 0.003
Specformer + TSE	0.888 ± 0.020	0.849 ± 0.013	0.978 ± 0.005	0.881 ± 0.008
Specformer + ETOSA	0.917 ± 0.013	0.777 ± 0.011	0.979 ± 0.007	0.869 ± 0.043
Specformer + TSE + ETOSA	0.905 ± 0.008	0.820 ± 0.029	0.981 ± 0.006	0.872 ± 0.007

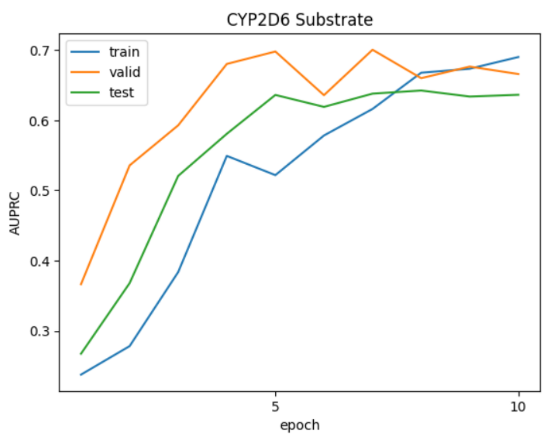
Table 2: Results on ADMET benchmark group. Shown is the mean ± s.d. for 5 runs.

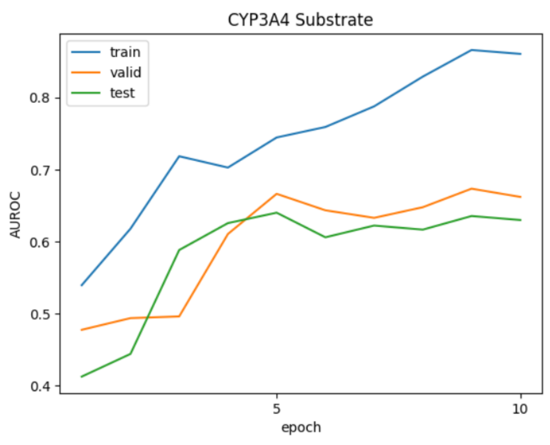
	Specformer with original third order self attention	Specformer with efficient third order self attention
AUROC	0.907 ± 0.014	0.917 ± 0.013
Computation time (s)	142	11
Memory Usage (MiB)	1566	1456
Number of parameters	341381	357765

Table 3: Results on DILI dataset. Shown is the mean ± s.d. for 5 runs.









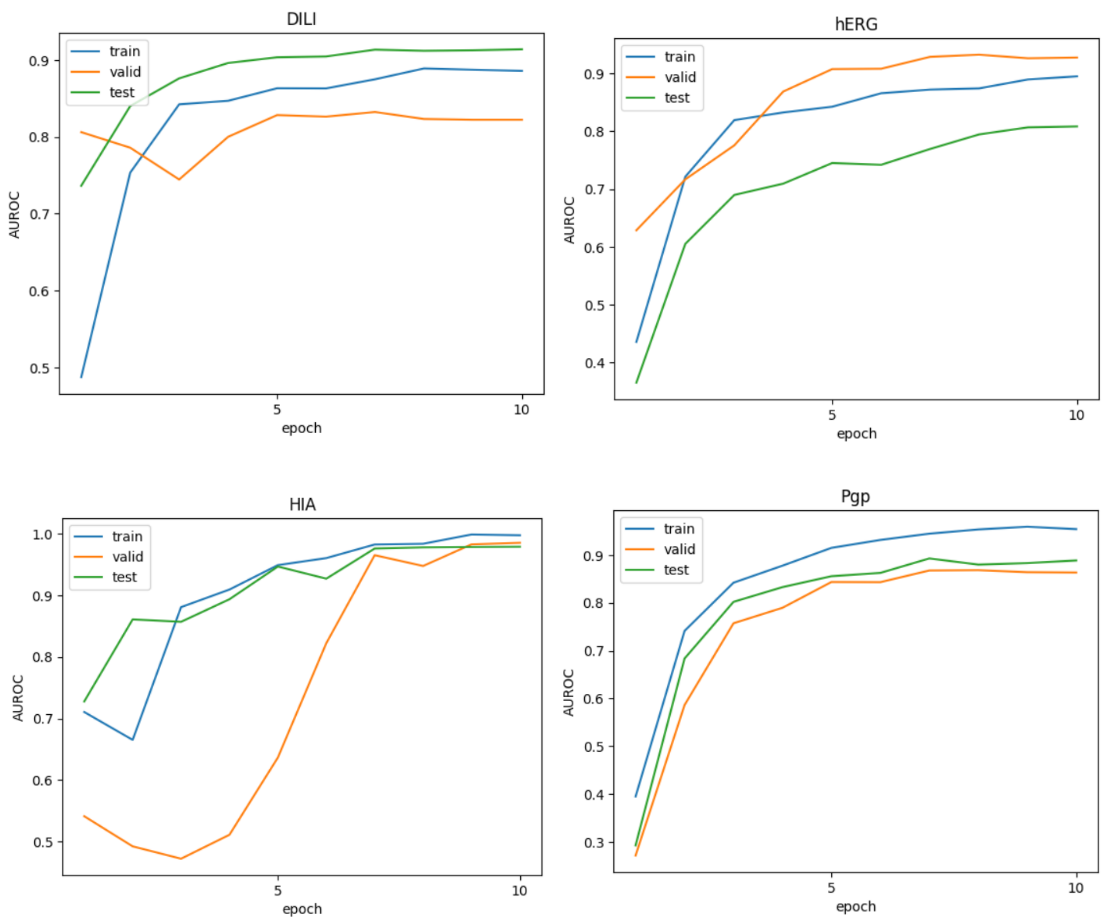


Figure: Performance of Specformer during training on ADMET benchmark group.

Section 9

Conclusion

We implemented a modification of a recently proposed higher order self attention using tensor operations efficiently, and applied it to a graph neural network with a novel structural encoding. Experiments on molecular graph datasets show that our modified higher order self attention is more efficient than the original higher order self attention, and our proposed structural encoding improves the performance of the graph neural network.

However, even though theoretically higher order self attention is more expressive than vanilla self attention, we observe that higher order self attention actually perform worse than the vanilla self attention on some datasets. One possible reason is that using higher order self attention layer may induce a bias in the model towards capturing higher order interactions, but not every dataset exhibits higher order interactions. A possible solution may be combining vanilla self attention with higher order self attention to counteract this bias.

Reference

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[2] Clayton Sanford, Daniel J Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers. Advances in Neural Information Processing Systems, 36, 2024.

[3] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. SIAM review, 51(3):455–500, 2009.

[4] Namgil Lee and Andrzej Cichocki. Fundamental tensor operations for large-scale data analysis in tensor train formats. arXiv preprint arXiv:1405.7786, 2014.

[5] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Lio, Guido F Montufar, and Michael Bronstein. Weisfeiler and lehman go cellular: Cw networks. Advances in neural information processing systems, 34:2625–2640, 2021.

[6] Mikhail Usvyatsov, Rafael Ballester-Ripoll, and Konrad Schindler. tntorch: Tensor network learning with PyTorch. Journal of Machine Learning Research, 23(208):1–6, 2022.

[7] Shanshan Tang, Bo Li, and Haijun Yu. Chebnet: Efficient and stable constructions of deep neural networks with rectified power units via chebyshev approximations. arXiv preprint arXiv:1911.05467, 2019.

[8] Mingguo He, Zhewei Wei, Hongteng Xu, et al. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. Advances in Neural Information Processing Systems, 34:14239–14251, 2021.

[9] Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In International Conference on Machine Learning, pages 23341–23362. PMLR, 2022.

[10] Deyu Bo, Chuan Shi, Lele Wang, and Renjie Liao. Specformer: Spectral graph neural networks meet transformers. arXiv preprint arXiv:2303.01028, 2023.

[11] Lorenzo Giusti, Claudio Battiloro, Lucia Testa, Paolo Di Lorenzo, Stefania Sardellitti, and Sergio Barbarossa. Cell attention networks. In 2023 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2023.

[12] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. arXiv preprint arXiv:2110.07875, 2021.

[13] Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor W Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. arXiv preprint arXiv:2102.09548, 2021.

[14] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101, 2017.

Appendix

Datasets

Name	Size	Task	Metric
Bioavailability	640	Classification	AUROC
CYP2C9 Substrate	666	Classification	AUPRC
CYP2D9 Substrate	664	Classification	AUPRC
CYP3A4 Substrate	667	Classification	AUROC
DILI	475	Classification	AUROC
hERG	648	Classification	AUROC
HIA	578	Classification	AUROC
Pgp	1212	Classification	AUROC

Table 3: Detailed information of datasets

Computational Resources All experiments are carried out on a cloud computing platform with:

- Operation system: Ubuntu 20.04
- CPU information: 18 vCPU AMD EPYC 9754 128-Core Processor
- GPU information: 1 GeForce RTX 4090D (24GB)

Data splitting For datasets from ADMET benchmark group, each dataset comes with a predefined test set containing 20% of the data. As suggested in [13], we adopt scaffold splitting to split the remaining data into a training set and a validation set with split ratio 70/10, and this procedure is repeated for 5 different random seeds to create 5 training sets and 5 validation sets.

Optimizer In all our experiments we used AdamW [14] optimizer, with the default settings of $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$, together with linear warm-up increase of the learning rate at the beginning of the training followed by its cosine decay.

Model Selection For datasets from ADMET benchmark group, we choose the last model checkpoint for evaluation.

Hyperparameters The hyperparameter settings are shown in Table 4 and Table 5.

Table 4: Hyperparameters for datasets from ADMET benchmark group

Hyperparameter	Bioavailability	CYP2C9 Substrate	CYP2D6 Substrate	CYP3A4 Substrate
Hidden dim	128	64	128	128
Layer	6	6	3	6
Head	4	4	4	4
Graph pooling	Mean	Mean	Mean	Mean
Architecture	Small	Small	Small	Small
Batch size	32	32	32	32
Epoch	10	10	10	10
Warmup	5	5	5	5
Learning rate	1e-4	1e-4	1e-4	1e-4
Weight decay	0	0	0	0
Transformer dropout	0.0	0.0	0.0	0.0
Feature dropout	0.0	0.0	0.0	0.0
Propagation dropout	0.0	0.0	0.0	0.0

Table 5: Hyperparameters for datasets from ADMET benchmark group

Hyperparameter	DILI	hERG	HIA	Pgp
Hidden dim	128	128	256	64
Layer	3	3	6	6
Head	4	4	4	4
Graph pooling	Mean	Mean	Mean	Mean
Architecture	Small	Small	Small	Small
Batch size	32	32	32	32
Epoch	10	10	10	10
Warmup	5	5	5	5
Learning rate	1e-4	1e-4	1e-4	1e-4
Weight decay	0	0	0	0
Transformer dropout	0.0	0.0	0.0	0.0
Feature dropout	0.0	0.0	0.0	0.0
Propagation dropout	0.0	0.0	0.0	0.0