Subgraphormer: Unifying Subgraph GNNs and Graph Transformers via Graph Products

Guy Bar-Shalom [1] **Beatrice Bevilacqua** [2] **Haggai Maron** [3 4]

Abstract

In the realm of Graph Neural Networks (GNNs), two exciting research directions have recently emerged: Subgraph GNNs and Graph Transformers. In this paper, we propose an architecture that integrates both approaches, dubbed `Subgraphormer`, which combines the enhanced expressive power, message-passing mechanisms, and aggregation schemes from Subgraph GNNs with attention and positional encodings, arguably the most important components in Graph Transformers. Our method is based on an intriguing new connection we reveal between Subgraph GNNs and product graphs, suggesting that Subgraph GNNs can be formulated as Message Passing Neural Networks (MPNNs) operating on a product of the graph with itself. We use this formulation to design our architecture: first, we devise an attention mechanism based on the connectivity of the product graph. Following this, we propose a novel and efficient positional encoding scheme for Subgraph GNNs, which we derive as a positional encoding for the product graph. Our experimental results demonstrate significant performance improvements over both Subgraph GNNs and Graph Transformers on a wide range of datasets.

1. Introduction

Due to their scalability and elegant architectural design, Message Passing Neural Networks (MPNNs) have become the most popular type of Graph Neural Networks (GNNs). Nonetheless, these architectures face significant limitations, particularly in terms of expressive power (Morris et al., 2019; Xu et al., 2018), resulting in underwhelming performances in certain tasks. Over the past few years, multiple GNN architectures have been proposed to mitigate these problems (Morris et al., 2021).

In this paper, we focus on two different enhanced GNN architecture families: Subgraph GNNs and Graph Transformers. In Subgraph GNNs (Zhang and Li, 2021; Cotta et al., 2021; Zhao et al., 2022; Bevilacqua et al., 2022; Frasca et al., 2022; Zhang et al., 2023b), a GNN is applied to a bag (multiset) of subgraphs, which is generated from the original graph (for example, the multiset of subgraphs obtained by deleting one node in the original graph). Notably, these architectures are provably more expressive than the traditional message passing algorithms applied directly to the original graph. In parallel, Transformers (Vaswani et al., 2017) have demonstrated outstanding performance across a wide range of applications, including natural language processing (Vaswani et al., 2017; Kalyan et al., 2021; Kitaev et al., 2020), computer vision (Khan et al., 2022; Dosovitskiy et al., 2020; Li et al., 2022b), and, more recently, graph-based tasks (Ying et al., 2021; Zhang et al., 2023b; Rampášek et al., 2022). Two of the most important components of Graph Transformers are their attention mechanism and their positional encoding scheme, which captures the graph structure. Graph Transformers have achieved impressive empirical performance, demonstrated by their state-of-the-art results on molecular tasks (Li et al., 2022a).

This work aims to integrate Subgraph GNNs and Graph Transformers into a unified architecture, which we call `Subgraphormer`, to leverage the benefits of both approaches. In order to define this hybrid approach, we develop two main techniques specifically adapted to subgraphs: (1) A subgraph attention mechanism, dubbed Subgraph Attention Block (SAB); and (2) A subgraph positional encoding scheme, which we call product graph PE. We derive these two mechanisms through a key observation: Subgraph GNNs can be naturally interpreted as MPNNs operating on a new product graph. This product graph is defined on 2-tuples of nodes and encodes both the subgraph structure and their inter-subgraph aggregation schemes.

Our approach builds on the recent maximally expressive

---

[1] Department of Computer Science, Technion - Israel Institute of Technology [2] Department of Computer Science, Purdue University [3] Department of Electrical & Computer Engineering, Technion - Israel Institute of Technology [4] NVIDIA Research. Correspondence to: Guy Bar-Shalom <guy.b@cs.technion.ac.il>.

(node-based) Subgraph GNN by Zhang et al. (2023a). The product graph of this architecture has two main types of edges. First, *internal* edges, that connect each vertex v in subgraph s to neighbors of v within s. Second, *external* edges, which are similarly constructed to connect v to copies of v itself across different subgraphs s' when the node generating subgraph s' is a neighbor of the node generating subgraph s. Those are illustrated in Section 3.1. These external edges enable the exchange of information between subgraphs, like the aggregation schemes used in existing Subgraph GNNs, which have been shown to improve performance and expressivity (Zhang et al., 2023a). Our SAB implements both internal and external attention-based aggregation methods, based on the edges defined above, thereby allowing individual nodes to refine their representations by selectively attending to nodes within the same or from different subgraphs. Importantly, as Subgraph GNNs are already computationally expensive, we restrict the attention to the product graph connectivity, similarly to sparse transformers (Choromanski et al., 2022; Shirzad et al., 2023).

Our Positional Encoding (PE) scheme, product graph PE, is tailored to the new product graph connectivity that emerges from the architecture defined above. Interestingly, this connectivity stems from the Cartesian product of the original graph with itself, which is a well-known type of graph product. We define our positional encoding as the Laplacian eigenvectors (Rampášek et al., 2022; Dwivedi et al., 2023; Kreuzer et al., 2021) of the adjacency matrix associated with this Cartesian product. Crucially, although the vertex set of the new graph comprises n^2 nodes, in contrast to n nodes in the original graph, the special structure of the Cartesian product graph allows us to compute positional encodings with time complexity equivalent to calculating positional encodings on the original, smaller graph. We demonstrate that those subgraph-positional encodings offer a performance boost for Subgraphormer, particularly in the case of larger graphs where stochastic subgraph sampling is used. We also describe how product graph PE, and Subgraphormer in general, can be generalized to other higher-order GNNs that operate on k-tuples of nodes.

An extensive experimental analysis over a variety of eight different datasets confirms that our architecture leads to significant performance improvements over both Subgraph GNNs and Graph Transformers, yielding outstanding results on multiple datasets, including ZINC-12K (Sterling and Irwin, 2015; Gómez-Bombarelli et al., 2018; Dwivedi et al., 2023) and the long-range benchmark PEPTIDES-STRUCT (Dwivedi et al., 2022). Furthermore, we address the potential computational burden of operating on large bags of subgraphs, demonstrating that the performance of stochastic bag sampling improves when using Subgraphormer compared to other Subgraph GNNs. In particular, we show that our positional encodings play a key role in this case.
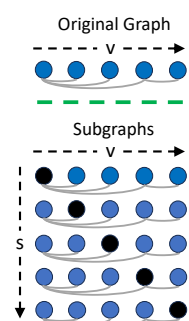

Figure 1: An example of generating subgraphs from the original graph. We denote by v the index used for the node dimension, and by s the one employed for the subgraph dimension. Each subgraph is generated by marking a single node in the original graph, with the marked node represented in black. We refer to the marked node as the root of the corresponding subgraph.

To summarize, the main contributions of this paper are (1) Subgraphormer, a novel architecture that combines the strengths of both transformer-based and subgraph-based architectures; (2) A novel observation connecting Subgraph GNNs to product graphs; (3) A novel positional encoding scheme tailored to subgraph methods; and (4) An empirical study demonstrating significant improvements of Subgraphormer compared to existing baselines in both full bag and stochastic bag sampling setups.

2. Related Work

Subgraph GNNs. Subgraph GNNs (Zhang and Li, 2021; Cotta et al., 2021; Papp et al., 2021; Bevilacqua et al., 2022; Zhao et al., 2022; Papp and Wattenhofer, 2022; Frasca et al., 2022; Qian et al., 2022; Huang et al., 2022; Zhang et al., 2023a; Bevilacqua et al., 2023) represent a graph as a collection of subgraphs, obtained by a predefined generation policy. For example, each subgraph can be generated by marking exactly one node in the original graph, an approach commonly referred to as *node marking* (Papp and Wattenhofer, 2022) – which is also the generation policy on which we focus in this paper. Since a node v of the original graph belongs to multiple subgraphs, we use the tuple (s, v) to refer to node v in subgraph s. Following the literature, we denote node s in subgraph s as the *root* of the subgraph. An example of this concept is illustrated in Figure 1. The success of Subgraph GNNs can be attributed to the enhanced expressive capabilities of these models, which surpass those of MPNNs working directly on the original graph.

Graph Transformers. Transformers have achieved notable success in natural language processing (Vaswani et al., 2017; Kalyan et al., 2021; Kitaev et al., 2020) and computer vision (Khan et al., 2022; Dosovitskiy et al., 2020; Li et al., 2022b), largely owing to their key component, the attention mechanism (Vaswani et al., 2017), often coupled with positional encoding schemes. Building on this, the GNN community has introduced Graph Transformers (Ying et al., 2021; Zhang et al., 2023b; Rampášek et al., 2022; Park et al., 2022; Chen et al., 2022), a specialized variant for graph-structured data. Graph Transformers incorporate either global attention (Kreuzer et al., 2021; Mialon et al.,

2021; Ying et al., 2021), which operates on fully-connected data, or make use of sparse attention (Shirzad et al., 2023; Choromanski et al., 2022; Brody et al., 2021), tailored to leverage the graph structure. Given the quadratic nature of Subgraph GNNs, we focus on sparse attention methods.

3. From Subgraph GNNs to Product Graphs

This section serves a dual purpose. Firstly, we demonstrate that the most expressive version of Subgraph GNNs (Zhang et al., 2023a), which serves as a main motivation in our paper, can be implemented by applying an MPNN to a newly constructed graph. Second, we explore the relationship between this new graph and the graph Cartesian product operation. Both of those aspects will be used in the following sections to develop our Subgraphormer architecture.

3.1. Subgraph GNNs as MPNNs

Notation. Let $G = (A, X)$ denote an undirected graph with node features.[1] The adjacency matrix $A \in \mathbb{R}^{n \times n}$ represents the graph connectivity while the feature matrix, $X \in \mathbb{R}^{n \times d}$, represents the node features. We denote the sets of nodes and edges as V and E, respectively, with $|V| = n$. Additionally, we denote neighbors in G by $\sim_G$, that is $v_1 \sim_G v_2$ denotes that v_1 and v_2 are neighbors in G. We denote the (usually highly sparse) adjacency and feature matrices of all subgraphs of a graph with calligraphic letters as, $\mathcal{A} \in \mathbb{R}^{n^2 \times n^2}$ and $\mathcal{X} \in \mathbb{R}^{n^2 \times d}$, respectively. Notably, we index $\mathcal{X}$ and $\mathcal{A}$ using the tuple (s, v), where $\mathcal{X}(s, v)$ is the feature of node v in subgraph s, and $\mathcal{A}((s, v), (s', v'))$ denotes the edge between node v in subgraph s and node v' in subgraph s'.

GNN-SSWL+. We build upon the construction of the maximally expressive Subgraph GNN proposed in Zhang et al. (2023a), which employs the *Node Marking* generation policy for generating the subgraphs, and updates the representation of node (s, v) according to the following formula:

$$
\mathcal{X}(s, v)^{t+1} = f^t\Big(\mathcal{X}(s, v)^t, \mathcal{X}(v, v)^t,
$$
$$
\{\mathcal{X}(s, v')^t\}_{v' \sim_G v}, \{\mathcal{X}(s', v)^t\}_{s' \sim_G s}, \Big), \quad (1)
$$

for an appropriate parameterized learnable function f^t, where the superscript t represents the layer number. Essentially, in each step, a node is updated using its representation, $\mathcal{X}(s, v)^t$, its representation in the subgraph it is root of, $\mathcal{X}(v, v)^t$, and two multisets of its neighbors' representations. The first multiset encompasses horizontal neighbors within the same subgraph, represented as $\{\mathcal{X}(s, v')^t\}_{v' \sim_G v}$, and the second multiset consists of vertical neighbors between the subgraphs, denoted by $\{\mathcal{X}(s', v)^t\}_{s' \sim_G s}$.

Product Graph Definition. In what follows, we construct an adjacency matrix of the form $\mathcal{A} \in \mathbb{R}^{n^2 \times n^2}$ to represent each one of the updates in Equation (1); we note that in all cases, this matrix, although in $\mathbb{R}^{n^2 \times n^2}$, is extremely sparse. We define the *product graph* to be the heterogeneous graph[2] defined by these adjacency matrices, together with a node feature matrix $\mathcal{X} \in \mathbb{R}^{n^2 \times d}$. Each adjacency we construct is visualized for the specific case where the original graph is the one given in Figure 1.

Internal subgraph connectivity. We start by introducing $\mathcal{A}_G \in \mathbb{R}^{n^2 \times n^2}$ (see inset on the left), which corresponds to $\{\mathcal{X}(s, v')^t\}_{v' \sim_G v}$ and maintains the connectivity of each subgraph separately. Formally,

$$
\mathcal{A}_G\Big((s, v), (s', v')\Big) = \begin{cases} \delta_{ss'} & \text{if } v \sim_G v'; \\ 0 & \text{otherwise,} \end{cases} \quad (2)
$$

where δ is the Kronecker delta.

External subgraph connectivity. The adjacency $\mathcal{A}_{G^S} \in \mathbb{R}^{n^2 \times n^2}$ corresponds to $\{\mathcal{X}(s', v)^t\}_{s' \sim_G s}$ (see inset) and contains edges across different subgraphs; hence denoted by the superscript S for G. In particular, these edges connect the same node in different subgraphs whenever the root nodes of these subgraphs are neighbors in the original graph. Formally,

$$
\mathcal{A}_{G^S}\Big((s, v), (s', v')\Big) = \begin{cases} \delta_{vv'} & \text{if } s \sim_G s'; \\ 0 & \text{otherwise.} \end{cases} \quad (3)
$$

Point-wise updates. The adjacency corresponding to $\mathcal{X}(v, v)^t$, denoted as $\mathcal{A}_{\text{point}}$, allows a node v in subgraph s to receive its root representation. Those updates are visualized inset by the grey arrows pointing from root nodes to other nodes in the same column. This adjacency matrix can be written as,

$$
\mathcal{A}_{\text{point}}\Big((s, v), (s', v')\Big) = \begin{cases} 1 & \text{if } s' = v' = v; \\ 0 & \text{otherwise.} \end{cases} \quad (4)
$$

To conclude our discussion on the relationship between GNN-SSWL+ and the product graph we constructed, we state and prove the following proposition showing that GNN-SSWL+ can be simulated by running an MPNN on the product graph:

---

[1]The consideration of edge features is omitted for simplicity.

[2]With a single node type and multiple edge types.

Proposition 3.1 (GNN-SSWL+ as an MPNN on the product graph). *Consider a graph $G = (A, X)$. Applying a stacking of RGCN layers (Schlichtkrull et al., 2018), interleaved with ReLU activations, on the product graph, as defined via the adjacencies in Equations (2) to (4), can implement the GNN-SSWL+ update in Equation (1).*

The proof is in Appendix E. This idea can be easily extended to other Subgraph GNN types. We note that the general idea of viewing GNNs as MPNNs operating on a new graph was recently proposed by Bause et al. (2023); Jogl et al. (2022; 2023), and a preliminary discussion on simulating Subgraph GNNs with MPNNs also appeared in Veličković (2022).

3.2. Connection to the Graph Cartesian Product

In the previous section, we described how to build a product graph for simulating a maximally expressive Subgraph GNN. Next we demonstrate that the connectivity of this particular product graph is tightly related to the concept of Graph Cartesian Product (Vizing, 1963; Harary, 2018).

Graph Cartesian Product. In simple terms, the Cartesian product of two graphs G_1 and G_2, denoted $G_1 \square G_2$, is a graph whose vertex set is the Cartesian product of $V(G_1)$ and $V(G_2)$, with two vertices (u_1, u_2) and (v_1, v_2) being adjacent if either $u_1 = v_1$ and u_2 is adjacent to v_2 in G_2, or $u_2 = v_2$ and u_1 is adjacent to v_1 in G_1; we denote the adjacency matrix of this new product graph as $\mathcal{A}_{G_1 \square G_2}$. In this paper, we mainly focus on a specific scenario where $G_1 = G_2$, namely, the Cartesian product of a graph with itself. For a formal definition, we refer the reader to Definition A.1 in Appendix A.

Given a graph $G = (A, X)$, the adjacency matrix corresponding to $G \square G$ can be expressed as:

$$\mathcal{A}_{G \square G} \triangleq A \otimes I + I \otimes A, \qquad (5)$$

with $\otimes$ the Kronecker product and I the identity matrix. We can now establish a direct connection between the product graph we have built for GNN-SSWL+ in the previous subsection and the Cartesian product graph.

Proposition 3.2 (Internal and External Connectivities give rise to the Cartesian Product Graph). *The edges induced by the internal and external subgraph connectivities, represented by $\mathcal{A}_G$ and $\mathcal{A}_{G^s}$ (Equations (2) and (3)) represent the connectivity of $\mathcal{A}_{G \square G}$. This implies the relationship:*

$$\mathcal{A}_{G \square G} = \mathcal{A}_{G^s} + \mathcal{A}_G. \qquad (6)$$

In particular, we have $\mathcal{A}_{G^s} = A \otimes I, \mathcal{A}_G = I \otimes A$.

The proof is given in Appendix E.

General Subgraph GNNs as Product Graphs. Different Subgraph GNNs differ in their aggregation schemes. While the aggregations defined by the adjacencies in Equations (2) to (4) are sufficient for maximal expressivity among known subgraph architectures, additional aggregation schemes, that may be important for certain tasks, have been proposed (Frasca et al., 2022). For instance, global updates enable nodes to refine their representations by incorporating information from other nodes, irrespective of their connectivity in the original graph. For example, a node (s, v), can aggregate information from all nodes within the same subgraph, (s, v'), for each $v' \in V$. Similarly, it can refine its representation by considering its copies in all subgraphs, (s', v), for each $s' \in V$. Both of these aggregations can be derived from the adjacency matrices of the Cartesian product $G_c \square G_c$. Here, G_c denotes the *clique* graph, whose adjacency matrix is $\vec{1}\vec{1}^T - I$. For a comprehensive discussion on the connection between Subgraph GNNs and Cartesian product graphs, refer to Appendix A.

4. Subgraphormer

Having established the relationship between Subgraph GNNs and product graphs, we can now use it to define our architecture: Subgraphormer. Subgraphormer is composed of a subgraph-specific positional encoding layer, node marking, a stacking of Subgraph Attention Blocks (SABs), and a final pooling layer, as depicted in Figure 2. In this section, we delve into the specifics of these components.

4.1. Subgraph Attention Block

In this subsection, we introduce the structure of the Subgraph Attention Block (SAB), designed to update the representation of each node v in subgraph s, denoted by $\mathcal{X}(s, v)$. In the SAB, the adjacency matrices defined above serve as sparsification functions, effectively biasing the attention mechanism to focus on the neighbors of each node in the product graph.

Implementing internal and external subgraph attentions. We implement two attention mechanisms - dubbed internal and external subgraph attention. These mechanisms are governed by the adjacency matrices $\mathcal{A}_G$ and $\mathcal{A}_{G^s}$, respectively. In general, we employ a sparse attention approach that computes Query ($\mathcal{Q}^t$), Key ($\mathcal{K}^t$), and Value ($\mathcal{V}^t$) transformations from the node features $\mathcal{X}^t \in \mathbb{R}^{n^2 \times d_1}$, according to the following equations:

$$\mathcal{Q}^t = \mathcal{X}^t \mathbf{W}_{\mathcal{Q}}^t, \ \mathcal{K}^t = \mathcal{X}^t \mathbf{W}_{\mathcal{K}}^t, \ \mathcal{V}^t = \mathcal{X}^t \mathbf{W}_{\mathcal{V}}^t, \quad (7)$$

where $\mathbf{W}_{\mathcal{Q}}^t, \mathbf{W}_{\mathcal{K}}^t, \mathbf{W}_{\mathcal{V}}^t \in \mathbb{R}^{d_1 \times d_2}$ are learned linear projections. The attention weights $\alpha^t(\mathcal{Q}^t, \mathcal{K}^t | \mathcal{A}) \in \mathbb{R}^{n^2 \times n^2}$ are computed from $\mathcal{Q}^t$ and $\mathcal{K}^t$ (e.g., by computing their outer product) *only* for the non-zero entries of the adjacency matrix $\mathcal{A} \in \{\mathcal{A}_G, \mathcal{A}_{G^s}\}$. The resulting node representation
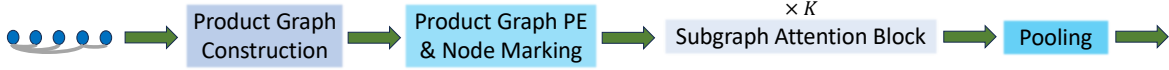
Figure 2: An overview of `Subgraphormer`. Given an input graph, we first construct the product graph and compute a subgraph-specific positional encoding. Then, we apply a stacking of Subgraph Attention Blocks, followed by a pooling layer to obtain a graph representation. A more comprehensive depiction of this figure can be found in Figure 3 of Appendix F.

are then computed employing the following equations:

$$\alpha^t_{\mathcal{A}_G}(\mathcal{Q}^t_{\mathcal{A}_G}, \mathcal{K}^t_{\mathcal{A}_G}|\mathcal{A}_G)\mathcal{V}^t_{\mathcal{A}_G}, \alpha^t_{\mathcal{A}_{G^S}}(\mathcal{Q}^t_{\mathcal{A}_{G^S}}, \mathcal{K}^t_{\mathcal{A}_{G^S}}|\mathcal{A}_{G^S})\mathcal{V}^t_{\mathcal{A}_{G^S}},$$

where we use different parameters (denoted by subscripts) to compute the attention coefficients of $\mathcal{A}_G, \mathcal{A}_{G^S}$. Importantly, our formulation of Subgraph GNNs as MPNNs on a product graph (Section 3) allows us to leverage existing optimized attention-based MPNNs to implement the `Subgraphormer` architecture. This avoids the need to build complex custom Subgraph GNN models from scratch. In particular, our implementation follows the Graph Attention Network (GAT) approach proposed by Veličković et al. (2017), which is implemented efficiently by accounting only for the non-zero entries in $\alpha(Q^t, K^t|\mathcal{A})$. We note that our approach can leverage any sparse attention model.

Implementing pointwise updates. For pointwise updates we do not use any attention mechanism, since every node has only a single neighbor. To additionally include the "self" (self loop) component from Equation (1) in this update, namely, $\mathcal{X}(s,v)$, we follow Zhang et al. (2023a) and employ a GIN encoder (Xu et al., 2018), that is,

$$\text{point}^t(\mathcal{X}^t, \mathcal{A}_{\text{Point}}) = \text{MLP}^t((1+\epsilon^t)\mathcal{X}^t + (\mathcal{A}_{\text{point}}\mathcal{X}^t)). \quad (8)$$

Final update step. Putting it all together, a single SAB is defined as an application of an MLP (applied to the feature dimension) to the concatenation of all the updates:

$$\text{SAB}^t(\mathcal{X}^t, \mathcal{A}_G, \mathcal{A}_{G^S}, \mathcal{A}_{\text{Point}}) =$$
$$\text{MLP}^t(\alpha^t_{\mathcal{A}_G}(\mathcal{Q}^t_{\mathcal{A}_G}, \mathcal{K}^t_{\mathcal{A}_G}|\mathcal{A}_G)\mathcal{V}^t_{\mathcal{A}_G} \oplus$$
$$\alpha^t_{\mathcal{A}_{G^S}}(\mathcal{Q}^t_{\mathcal{A}_{G^S}}, \mathcal{K}^t_{\mathcal{A}_{G^S}}|\mathcal{A}_{G^S})\mathcal{V}^t_{\mathcal{A}_{G^S}} \oplus \text{point}^t(\mathcal{X}^t, \mathcal{A}_{\text{Point}})),$$

where $\oplus$ represents feature concatenation.

We remark that one way of looking at a SAB update is by taking inspiration from Proposition 3.1 and extending the message passing within RGCN to further include different attention weights for each edge type, enabling nodes to attend differently to nodes connected via various edge types.

Pooling. After a stacking of SAB layers, we employ a pooling layer to obtain a graph representation, that is,

$$\rho(\mathcal{X}^T) = \text{MLP}^T\Big(\sum_{s=1}^n \Big(\sum_{v=1}^n \mathcal{X}^T(s,v)\Big)\Big), \quad (9)$$

where T denotes the final layer.

We note that a stacking of SABs followed by an invariant pooling layer as described above, guarantees invariance to node permutations. This is because each block in our model maintains equivariance to node permutations, while the final pooling layer ensures invariance to such permutations.

Complexity. SAB matches GNN-SSWL+ in time and space complexity, with both at $\mathcal{O}(|V|^2 + |V||E|)$ for an original graph of $|V|$ nodes and $|E|$ edges. See Appendix D for a detailed complexity analysis of each aggregation.

4.2. Subgraph Positional Encodings

Positional encodings for graphs have been well studied (Dwivedi et al., 2021; Wang et al., 2022; Lim et al., 2022), and were shown to provide valuable information for both message-passing based GNNs and graph transformers.

A prominent approach that we follow in this work makes use of the eigendecomposition of the graph Laplacian matrix. More specifically, given a graph $G = (A, X)$, the Laplacian is defined as $L = D - A$, where $D = \text{diag}(A\vec{1})$ is the degree matrix. The positional encoding is calculated by an eigendecomposition of the Laplacian, $L = U^T \Lambda U$, as

$$\mathbf{p}_v^{:k} \triangleq [U_{v1}, \dots, U_{vk}], \quad (10)$$

as the encoding for node v, where k is a hyperparameter denoting the number of eigenvectors we consider, $k \leq n$. We note that the eigenvectors are sorted (in ascending order) according to their eigenvalues.

Unfortunately, directly applying this approach to Subgraph GNNs is not straightforward for two reasons: *(1) Unclear adjacency structure:* For the Laplacian to be computed, it is necessary to determine the relevant symmetric adjacency matrix. Specifically, Subgraph GNNs, in general, operate over a collection of subgraphs rather than on a graph structure. It is not clear what type of connectivity we should use for the Laplacian of this structure. *(2) Efficiency concerns:* The adjacency matrix for Subgraph GNNs is of size $n^2 \times n^2$. Consequently, computing its Laplacian's k eigenvectors results in a computational complexity of $\mathcal{O}(k \cdot n^4)$, as opposed to $\mathcal{O}(k \cdot n^2)$(Lanczos, 1950; Lehoucq et al., 1998; Lee et al., 2009) for a graph with n nodes, rendering it impractical in many scenarios.

In the following, we design a positional encoding scheme that addresses these two challenges. We start with the adjacency challenge. A natural solution to (1) is to employ

the adjacencies $\mathcal{A}_G$, $\mathcal{A}_{G^s}$ in Equations (2) and (3), excluding $\mathcal{A}_{\text{point}}$ since it is asymmetrical and the connectivity it represents is not related to the original graph. These adjacencies correspond to the main connectivity employed by `Subgraphormer`. Consequently, our proposed positional encoding method will primarily focus on the symmetric connectivity defined by $\mathcal{A}_G$ and $\mathcal{A}_{G^s}$ only. This choice brings us to the challenge (2).

To address point (2), we utilize Proposition 3.2 that established a relationship between the adjacency matrices $\mathcal{A}_{G^s}$, $\mathcal{A}_G$, and the Cartesian product graph, expressed as,

$$\mathcal{A}_{G\square G} = \mathcal{A}_{G^s} + \mathcal{A}_G = A \otimes I + I \otimes A. \quad (11)$$

Surprisingly, Equation (11) simplifies the computation of eigenvectors for the Laplacian $\mathcal{L}_{G\square G}$, by leveraging the eigenvectors and eigenvalues of the original graph G, as detailed in the following proposition (Barik et al., 2015).

Proposition 4.1 (Product Graph eigendecomposition). *Consider a graph $G = (A, X)$.[3] The eigenvectors and eigenvalues of the Laplacian matrix of $G\square G$, namely, $\mathcal{L}_{G\square G}$, are $\{(v_i \otimes v_j, \lambda_i + \lambda_j)\}_{i,j=1}^{n^2}$, where $\{(v_i, \lambda_i)\}_{i=1}^{n}$ are the eigenvectors and eigenvalues of the Laplacian matrix of G.*

We refer to Appendix E for the proof. The implications of this observation are profound. It reveals that, despite the fact that `Subgraphormer` processes product graphs with n^2 nodes, computing the positional encodings only requires an eigendecomposition of the original smaller graph. Specifically, for an undirected graph $G = (A, X)$ with n vertices and its Cartesian product graph $G\square G$, calculating the first k eigenvectors, $k \leq n$, has a time complexity of $\mathcal{O}(k \cdot n^2)$ (Lanczos, 1950; Lee et al., 2009) – the same as computing k eigenvectors for the Laplacian of G. A proof for this claim is also given Appendix E. We refer to our subgraph-positional encodings as *product graph PE*, or simply *PE*.

Visualizing product graph PE. To illustrate the benefits of our product graph PE, we visualize in different colors the entries of the first non-trivial eigenvector of both the original graph (upper) and the product graph (lower).
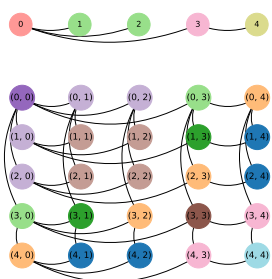
It is readily apparent that the eigenvector on the product graph conveys more information than the one on the original graph. This distinction is particularly evident by observing the color diversity, with the product graph featuring ten distinct colors as opposed to the four in the original graph.

Concatenation PE. We briefly introduce another subgraph positional encoding method, *concatenation PE*. Given the

Laplacian matrix $L = U^T \Lambda U$, of a graph G, the concatenation PE for a node (s, v) is given by the output of an `MLP` on the concatenated vectors $\mathbf{p}_s^{:k} = [U_{s1}, \ldots, U_{sk}]$ and $\mathbf{p}_v^{:k} = [U_{v1}, \ldots, U_{vk}]$; we highlight that those two vectors corresponds to the eigenvectors when considering each of the connectivities, $\mathcal{A}_G, \mathcal{A}_{G^s}$, independently. Notably, concatenation PE can approximate (up to an ordering) the product graph PE. This is because the product graph PE multiplies specific elements from the concatenation PE, a process that can be approximated using a `MLP` by using the Universal Approximation Theorem (Hornik, 1991; Cybenko, 1989). However, we use the product graph PE in this study due to its superior performance, as shown in the comparison over the ZINC-12K dataset in Appendix C.3.1.

Node Marking. Motivated by existing Subgraph GNNs, we leverage an additional "special mark", defined as *node marking*, which we concatenate to each node's positional encoding. Following the approach of Zhang et al. (2023a), we hold a lookup table, and assign a learnable embedding to each node, represented as $z_{\text{dist}(s,v)} \in \mathbb{R}^d$. The embedding assigned to node (s, v) is indexed by the shortest path distance between them in the original graph G – $\text{dist}(s, v)$.[4]

4.3. Scaling up `Subgraphormer`

Given a graph G with n nodes, `Subgraphormer` operates on a transformed graph with n^2 nodes. Even though this transformed graph is extremely sparse, this is a well-known drawback of Subgraph GNNs (Qian et al., 2022; Kong et al., 2023; Bevilacqua et al., 2023) when considering large graphs, since processing the new graph might be infeasible.
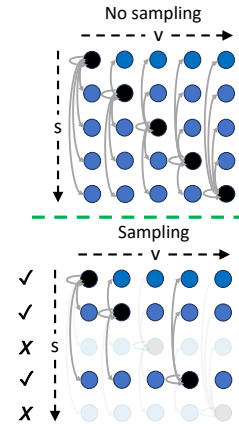
Thus, in this section we adopt the idea of stochastic sampling (Bevilacqua et al., 2022; Cotta et al., 2021; Zhao et al., 2022). Specifically, we adapt `Subgraphormer` to stochastic sampling by adding sparsifications to the adjacencies based on the sampling (see inset). We sample rows (subgraphs) uniformly at random and include an edge if and only if both of its endpoints are part of the sampled graph.

Formally, we introduce a sampling mask, $\mathcal{A}_{\text{mask}}$, defined as,

$$\mathcal{A}_{\text{mask}}\Big((s, v), (s', v')\Big) = \begin{cases} 1 & \text{if } s \text{ and } s' \text{ are sampled}; \\ 0 & \text{otherwise}. \end{cases}$$

Given an adjacency matrix $\mathcal{A}$ and a generated sampling mask $\mathcal{A}_{\text{mask}}$, we perform an element-wise logical AND oper-

---

[3]We assume A has no self loops.

[4]We assign a unique mark if the two nodes are unreachable from each other, i.e., $\text{dist}(s, v) = \infty$.

ation between them. Consequently, $\mathcal{A}$ is adjusted to preserve only edges that connect node pairs in sampled subgraphs.

Importantly, our *product graph PE* (Section 4.2) can be computed independently from the sampling process. This feature allows them to retain information from the complete, unsampled graph. As demonstrated in the experimental section, this attribute of *product graph PE* proves advantageous, as it appears to effectively compensate for the information loss incurred due to sampling.

4.4. Generalization to Higher-Order Graph Networks

While our current derivation considers product graphs containing 2-tuples of nodes, both the SAB and the product graph PE can be naturally extended to define similar high-order architectures operating on k-tuples of nodes (Morris et al., 2019; Maron et al., 2018). This extension is grounded in our understanding that Subgraph GNNs are connected to the Cartesian product of graphs. Accordingly, this product can be generalized to be applied k times, rather than being limited to $k = 2$. This generalization leads to the formation of k distinct adjacency matrices, each representing different "Internal/External" aggregations. To elaborate, the i-th adjacency matrix is defined as $\underbrace{I \otimes I \otimes \cdots \otimes I \otimes A \otimes I \otimes \cdots \otimes I \otimes I}_{\text{with } A \text{ exclusively positioned in the } i\text{-th slot}}$. Similarly, the positional encodings can also be extended to any k-tuple, with the eigenvectors obtained by the tensor product of k eigenvectors of G, and the eigenvalues by the sum of the respective eigenvalues. More specifically, for each sequence of indices $\{i_1, i_2, \ldots, i_k\}$, where $i_j \in \{1, 2, \ldots, n\}$ for each j, there exists an eigenvector-eigenvalue pair given by $(v_{i_1} \otimes v_{i_2} \otimes \ldots \otimes v_{i_k}, \lambda_{i_1} + \lambda_{i_2} + \ldots + \lambda_{i_k})$. Here, $\{(v_i, \lambda_i)\}_{i=1}^{n}$ represent the eigenvectors and eigenvalues of the Laplacian matrix of G. Appendix B presents a detailed derivation of SABs and PEs for k-tuples, for any given k.

Additionally, we note that the high-order positional encoding discussed here may be useful for other high-order GNNs.

5. Experiments

We conducted an extensive set of experiments over eight different datasets to answer the following questions: (Q1) *Can Subgraphormer outperform Subgraph GNNs and Graph Transformers in real-world benchmarks?* (Q2) *Does the attention mechanism prove advantageous?* (Q3) *Does our product graph positional encoding scheme boost performance, and in which contexts?* (Q4) *How well does Subgraphormer perform when using stochastic sampling?* (Q5) *Can Subgraphormer offer competitive performance on long-range benchmarks when compared to Graph Transformers?*

All experiments were conducted using PyTorch, and the

Table 1: On the ZINC datasets, Subgraphormer outperforms Graph Transformers and Subgraph GNNs . The top three results are reported as **First**, **Second**, and **Third**.

Model ↓ / Dataset →	Param.	ZINC-12K (MAE ↓)	ZINC-FULL (MAE ↓)
GSN (Bouritsas et al., 2022)	500k	0.101±0.010	-
CIN (small) (Bodnar et al., 2021)	100k	0.094±0.004	0.044±0.003
GIN (Xu et al., 2018)	500k	0.163±0.004	-
PPGN++(6) (Puny et al., 2023)	500k	0.071±0.001	**0.020**±0.001
SAN (Kreuzer et al., 2021)	509k	0.139±0.006	-
URPE (Luo et al., 2022)	492k	0.086±0.007	0.028±0.002
GPS (Rampášek et al., 2022)	424k	**0.070**±0.004	-
Graphormer (Ying et al., 2021)	489k	0.122±0.006	0.052±0.005
Graphormer-GD (Zhang et al., 2023b)	503k	0.081±0.009	0.025±0.004
K-Subgraph SAT (Chen et al., 2022)	523k	0.094±0.008	-
NGNN (Zhang and Li, 2021)	500k	0.111±0.003	0.029±0.001
DS-GNN (Bevilacqua et al., 2022)	100k	0.116±0.009	-
DSS-GNN (Bevilacqua et al., 2022)	100k	0.102±0.003	0.029±0.003
GNN-AK (Zhao et al., 2022)	500k	0.105±0.010	-
GNN-AK+ (Zhao et al., 2022)	500k	0.091±0.002	-
SUN (Frasca et al., 2022)	526k	0.083±0.003	0.024±0.003
OSAN (Qian et al., 2022)	500k	0.154±0.008	-
DS-GNN (Bevilacqua et al., 2023)	500k	0.087±0.003	-
GNN-SSWL (Zhang et al., 2023a)	274k	0.082±0.003	0.026±0.001
GNN-SSWL+ (Zhang et al., 2023a)	387k	**0.070**±0.005	**0.022**±0.001
Subgraphormer	293k	**0.067**±0.007	**0.020**±0.002
Subgraphormer + PE	293k	**0.063**±0.001	**0.023**±0.001

Table 2: On the OGB datasets, Subgraphormer outperforms Graph Transformers and Subgraph GNNs . The top three results are reported as **First**, **Second**, and **Third**.

Model ↓ / Dataset →	MOLHIV (ROC-AUC ↑)	MOLBACE (ROC-AUC ↑)	MOLESOL (RMSE ↓)
GCN (Kipf and Welling, 2016)	76.06±0.97	**79.15**±1.44	1.114±0.036
GIN (Xu et al., 2018)	75.58±1.40	72.97±4.00	1.173±0.057
PNA (Corso et al., 2020)	79.05±1.32	-	-
GSN (Bouritsas et al., 2022)	**80.39**±0.90	-	-
CIN (Bodnar et al., 2021)	**80.94**±0.57	-	-
SAN (Kreuzer et al., 2021)	77.85±0.24	-	-
GPS (Rampášek et al., 2022)	78.80±1.01	-	-
NGNN (Zhang and Li, 2021)	78.34±1.86	-	-
Recon. GNN (Cotta et al., 2021)	76.32±1.40	1.026±0.033	-
DS-GNN (Bevilacqua et al., 2022)	77.40±2.19	-	-
DSS-GNN (Bevilacqua et al., 2022)	76.78±1.66	-	-
GNN-AK+ (Zhao et al., 2022)	79.61±1.19	-	-
SUN (Frasca et al., 2022)	80.03±0.55	-	-
OSAN (Qian et al., 2022)	-	72.30±6.60	0.980±0.086
DS-GNN (Bevilacqua et al., 2023)	76.54±1.37	78.41±1.94	0.847±0.015
GNN-SSWL+ (Zhang et al., 2023a)	79.58±0.35	**82.70**±1.80	**0.837**±0.019
Subgraphormer	**80.38**±1.92	**81.62**±3.55	**0.832**±0.043
Subgraphormer + PE	79.48±1.28	**84.35**±0.65	**0.826**±0.010

corresponding code is publicly available for replicating our results[5].

In the following, we present our main results and refer to Appendix C for additional experiments.

Real-world data. On the ZINC-12K and ZINC-FULL datasets (Sterling and Irwin, 2015; Gómez-Bombarelli et al., 2018; Dwivedi et al., 2023), Subgraphormer outperforms both Subgraph GNNs and Graph Transformers by a

---

[5]https://github.com/BarSGuy/Subgraphormer

Table 3: Ablation Study over our product graph PE. For each dataset, the best result for each sampling ratio is in **Bold**.

Dataset ↓ / Sampling Ratio →		Subgraphormer / Subgraphormer + PE						
		100%		50%		20%		5%
ZINC-12K	(MAE ↓)	0.067±0.007	**0.063**±0.001	0.084±0.002	**0.084**±0.002	0.121±0.007	**0.120**±0.002	0.200±0.017	**0.175**±0.006
MOLHIV	(ROC-AUC ↑)	**80.38**±1.92	79.48±1.28	**79.66**±0.79	79.61±1.30	76.48±2.38	**76.68**±1.07	70.87±0.90	**71.02**±0.79
MOLBACE	(ROC-AUC ↑)	81.62±3.55	**84.35**±0.65	79.49±2.38	**83.82**±2.62	75.27±5.63	**78.77**±4.10	63.05±8.70	**67.73**±5.50
MOLESOL	(RMSE ↓)	0.832±0.043	**0.826**±0.010	0.829±0.013	**0.812**±0.001	1.093±0.009	**1.041**±0.030	**1.266**±0.019	1.270±0.007

Table 4: Results on the PEPTIDES datasets demonstrate the effectiveness of the stochastic variant of Subgraphormer in tasks where Graph Transformers excel. Graph Transformers and Subgraph GNNs are highlighted in gray and light blue. The top three results are reported as **First**, **Second**, and **Third**.

Model ↓ / Dataset →		PEPTIDES-FUNC (AP ↑)	PEPTIDES-STRUCT (MAE ↓)
GCN (Kipf and Welling, 2016)		0.5930±0.0023	0.3496±0.0013
GIN (Xu et al., 2018)		0.5498±0.0079	0.3547±0.0045
GatedGCN (Bresson and Laurent, 2017)		0.5864±0.0077	0.3420±0.0013
GatedGCN+RWSE (Dwivedi et al., 2022)		0.6069±0.0035	0.3357±0.0006
Transf.+LapPE (Dwivedi et al., 2022)		0.6326±0.0126	0.2529±0.0016
SAN+LapPE (Kreuzer et al., 2021)		0.6384±0.0121	0.2683±0.0043
SAN+RWSE (Kreuzer et al., 2021)		**0.6439**±0.0075	0.2545±0.0012
GPS (Rampášek et al., 2022)		**0.6535**±0.0041	**0.2500**±0.0005
GNN-SSWL+ (Zhang et al., 2023a)	30%	0.5847±0.0050	0.2570±0.0006
Subgraphormer	30%	**0.6415**±0.0052	**0.2494**±0.0020
Subgraphormer + PE	30%	0.6373±0.0110	**0.2475**±0.0007

significant margin, while using a smaller number of parameters (Table 1). Importantly, the attention mechanism proves advantageous, as Subgraphormer always outperforms GNN-SSWL+ (Zhang et al., 2023a), on which it builds on by integrating attention blocks. Remarkably, our product graph PE boosts the performance of Subgraphormer on ZINC-12K, where it is the top-performing method. A similar trend is observed on the OGB datasets (Hu et al., 2020) in Table 2. Notably, on the MOLBACE dataset, the inclusion of the PE improves the performance by 3%. Additional results on ALCHEMY-12K are reported in Appendix C.3.

Efficacy of stochastic sampling on long-range data. We experimented on the PEPTIDES-FUNC and PEPTIDES-STRUCT datasets (Dwivedi et al., 2022), evaluating the ability of Subgraphormer to scale to larger graphs and to capture long-range dependencies, a notoriously hard task for MPNNs (Dwivedi et al., 2022). We considered a sampling ratio of 30%, which allows us to run Subgraphormer though typical full-bag Subgraph GNNs cannot be applied. The results are summarized in Table 4. Despite the reduced number of subgraphs, Subgraphormer achieves the best performance on PEPTIDES-STRUCT, outperforming all Graph Transformers, which excel on tasks of this kind, and offer comparable results over PEPTIDES-FUNC.

Ablation Study: Product Graph PE. We assess the impact of our product graph PE on the performance of the stochastic variant of Subgraphormer under common sampling rates (Bevilacqua et al., 2022) (100%, 50%, 20%, 5%) across four datasets. Table 3 shows that the inclusion of our product graph PE improves the results of Subgraphormer in 13 out of 16 dataset-sampling combinations. Its effectiveness is especially noticeable when combined with 20% and 5% ratios, indicating its role in compensating information loss due to low sampling rates.

Discussion. In what follows, we address research questions **Q1** to **Q5**.

A1. Tables 1 and 2 show Subgraphormer outperforming all transformer-based and subgraph-based baselines. A similar trend is observed on the ALCHEMY-12K dataset in Table 7 in Appendix C.3.

A2. Tables 1 and 4 clearly demonstrate the importance of the attention, as Subgraphormer always outperforms GNN-SSWL+, from which it is built by adding SABs. On the OGB (Table 2) the attention mechanism proves particularly beneficial when coupled with the product graph PE.

A3. Table 3 demonstrates the effectiveness of the PE, especially when considering lower sampling regimes.

A4. The stochastic variant of Subgraphormer proves particularly advantageous in settings where full-bag Subgraph GNNs cannot be otherwise applied, such as in Table 4. Additional results can be found in Tables 8 to 10 in Appendix C.

A5. Table 4 demonstrates the ability of Subgraphormer to capture long-range dependencies, as it outperforms all baselines on the PEPTIDES-STRUCT dataset, and performs comparably over PEPTIDES-FUNC.

6. Conclusions

In this work, we introduce Subgraphormer, a novel architecture that merges the capabilities of Subgraph GNNs and Graph Transformers. Building upon our observation that views Subgraph GNNs as MPNNs operating on product graphs, we propose: (1) a subgraph-based attention mechanism, which can be implemented by leveraging optimized sparse attention blocks operating on the product

graph; and (2) a positional encoding scheme capturing the connectivity of the product graph computed in a time complexity equivalent to computing positional encodings on the original, small graph. Empirically, we demonstrate that `Subgraphormer` outperforms Subgraph GNNs and Graph Transformers over a wide range of datasets. We also investigated the capabilities of the stochastic variant of our approach, demonstrating impressive performance on long-range datasets, where Graph Transformers excel, and full-bag Subgraph GNNs cannot be applied.

Acknowledgements

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Sasmita Barik, Ravindra B Bapat, and Sukanta Pati. On the laplacian spectra of product graphs. *Applicable Analysis and Discrete Mathematics*, pages 39–58, 2015.

Franka Bause, Fabian Jogl, Patrick Indri, Tamara Drucks, David Penz, Nils Kriege, Thomas Gärtner, Pascal Welke, and Maximilian Thiessen. Maximally expressive gnns for outerplanar graphs. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023.

Beatrice Bevilacqua, Fabrizio Frasca, Derek Lim, Balasubramaniam Srinivasan, Chen Cai, Gopinath Balamurugan, Michael M Bronstein, and Haggai Maron. Equivariant subgraph aggregation networks. *International Conference on Learning Representations*, 2022.

Beatrice Bevilacqua, Moshe Eliasof, Eli Meirom, Bruno Ribeiro, and Haggai Maron. Efficient subgraph gnns by learning effective selection policies. *International Conference on Learning Representations*, 2023.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

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.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2022.

Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

Dexiong Chen, Leslie O'Bray, and Karsten Borgwardt. Structure-aware transformer for graph representation learning. In *International Conference on Machine Learning*, pages 3469–3489. PMLR, 2022.

Guangyong Chen, Pengfei Chen, Chang-Yu Hsieh, Chee-Kong Lee, Benben Liao, Renjie Liao, Weiwen Liu, Jiezhong Qiu, Qiming Sun, Jie Tang, et al. Alchemy: A quantum chemistry dataset for benchmarking ai models. *arXiv preprint arXiv:1906.09427*, 2019.

Krzysztof Choromanski, Han Lin, Haoxian Chen, Tianyi Zhang, Arijit Sehanobish, Valerii Likhosherstov, Jack Parker-Holder, Tamas Sarlos, Adrian Weller, and Thomas Weingarten. From block-toeplitz matrices to differential equations on graphs: towards a general theory for scalable masked transformers. In *International Conference on Machine Learning*, pages 3962–3983. PMLR, 2022.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. Reconstruction for powerful graph representations. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Mohammed Haroon Dupty and Wee Sun Lee. Graph representation learning with individualization and refinement. *arXiv preprint arXiv:2203.09141*, 2022.

Mohammed Haroon Dupty, Yanfei Dong, and Wee Sun Lee. Pf-gnn: Differentiable particle filtering based approximation of universal graph representations. In *International Conference on Learning Representations*, 2021.

Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. *International Conference on Learning Representations*, 2021.

Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, 35:22326–22340, 2022.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Fabrizio Frasca, Beatrice Bevilacqua, Michael Bronstein, and Haggai Maron. Understanding and extending subgraph gnns by rethinking their symmetries. *Advances in Neural Information Processing Systems*, 35:31376–31390, 2022.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Frank Harary. *Graph Theory (on Demand Printing Of 02787)*. CRC Press, 2018.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

Yinan Huang, Xingang Peng, Jianzhu Ma, and Muhan Zhang. Boosting the cycle counting power of graph neural networks with i^2-gnns. In *The Eleventh International Conference on Learning Representations*, 2022.

Fabian Jogl, Maximilian Thiessen, and Thomas Gärtner. Weisfeiler and leman return with graph transformations. In *18th International Workshop on Mining and Learning with Graphs*, 2022.

Fabian Jogl, Maximilian Thiessen, and Thomas Gärtner. Expressivity-preserving gnn simulation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv preprint arXiv:2108.05542*, 2021.

Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations*, 2016.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Lecheng Kong, Jiarui Feng, Hao Liu, Dacheng Tao, Yixin Chen, and Muhan Zhang. Mag-gnn: Reinforcement learning boosted graph neural network. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

Jungmin Kwon, Jeongseop Kim, Hyunseo Park, and In Kwon Choi. Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks. In *International Conference on Machine Learning*, 2021.

Cornelius Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators1. *Journal of Research of the National Bureau of Standards*, 45(4), 1950.

Jongwon Lee, Venkataramanan Balakrishnan, Cheng-Kok Koh, and Dan Jiao. From o (k 2 n) to o (n): A fast complex-valued eigenvalue solver for large-scale on-chip interconnect analysis. In *2009 IEEE MTT-S International Microwave Symposium Digest*, pages 181–184. IEEE, 2009.

Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK users' guide: solution of large-scale eigenvalue problems with implicitly restarted Arnoldi methods.* SIAM, 1998.

Han Li, Dan Zhao, and Jianyang Zeng. Kpgt: knowledge-guided pre-training of graph transformer for molecular property prediction. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 857–867, 2022a.

Yanghao Li, Chao-Yuan Wu, Haoqi Fan, Karttikeya Mangalam, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Mvitv2: Improved multiscale vision transformers for classification and detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4804–4814, 2022b.

Derek Lim, Joshua Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and basis invariant networks for spectral graph representation learning. *The Eleventh International Conference on Learning Representations*, 2022.

Shengjie Luo, Shanda Li, Shuxin Zheng, Tie-Yan Liu, Liwei Wang, and Di He. Your transformer may not be as powerful as you expect. *Advances in Neural Information Processing Systems*, 35:4301–4315, 2022.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*, 2018.

Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019.

Christopher Morris, Gaurav Rattan, and Petra Mutzel. Weisfeiler and leman go sparse: Towards scalable higher-order graph embeddings. *Advances in Neural Information Processing Systems*, 33:21824–21840, 2020.

Christopher Morris, Yaron Lipman, Haggai Maron, Bastian Rieck, Nils M Kriege, Martin Grohe, Matthias Fey, and Karsten Borgwardt. Weisfeiler and leman go machine learning: The story so far. *arXiv preprint arXiv:2112.09992*, 2021.

Christopher Morris, Gaurav Rattan, Sandra Kiefer, and Siamak Ravanbakhsh. Speqnets: Sparsity-aware permutation-equivariant graph networks. In *International*

Conference on Machine Learning, pages 16017–16042. PMLR, 2022.

Pál András Papp and Roger Wattenhofer. A theoretical comparison of graph neural network extensions. In *International Conference on Machine Learning*, pages 17323–17345. PMLR, 2022.

Pál András Papp, Karolis Martinkus, Lukas Faber, and Roger Wattenhofer. Dropgnn: Random dropouts increase the expressiveness of graph neural networks. *Advances in Neural Information Processing Systems*, 34:21997–22009, 2021.

Wonpyo Park, Woonggi Chang, Donggeon Lee, Juntae Kim, and Seung-won Hwang. Grpe: Relative positional encoding for graph transformer. *arXiv preprint arXiv:2201.12787*, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Omri Puny, Derek Lim, Bobak Kiani, Haggai Maron, and Yaron Lipman. Equivariant polynomials for graph neural networks. In *International Conference on Machine Learning*, pages 28191–28222. PMLR, 2023.

Chendi Qian, Gaurav Rattan, Floris Geerts, Mathias Niepert, and Christopher Morris. Ordered subgraph aggregation networks. *Advances in Neural Information Processing Systems*, 35:21030–21045, 2022.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Scaling graph transformers with expander graphs. *International Conference on Machine Learning*, 2023.

Teague Sterling and John J Irwin. Zinc 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

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.

Petar Veličković. Message passing all the way up. *arXiv preprint arXiv:2202.11097*, 2022.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *International Conference on Learning Representations*, 2017.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Vladim G Vizing. The cartesian product of graphs. *Vycisl. Sistemy*, 9(30-43):33, 1963.

Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. *International Conference on Learning Representations*, 2022.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *International Conference on Learning Representations*, 2018.

Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Small relu networks are powerful memorizers: a tight analysis of memorization capacity. *Advances in Neural Information Processing Systems*, 32, 2019.

Bohang Zhang, Guhao Feng, Yiheng Du, Di He, and Liwei Wang. A complete expressiveness hierarchy for subgraph gnns via subgraph weisfeiler-lehman tests. *International Conference on Machine Learning*, 2023a.

Bohang Zhang, Shengjie Luo, Liwei Wang, and Di He. Rethinking the expressive power of gnns via graph biconnectivity. *International Conference on Learning Representations*, 2023b.

Muhan Zhang and Pan Li. Nested graph neural networks. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any GNN with local structure awareness. In *International Conference on Learning Representations*, 2022.

A. Subgraph GNNs as Cartesian Product Graphs

In this section, we delve into the relationship between the Cartesian product graph and `Subgraphormer` (Appendix A.1). Additionally, we explore the broader connection between Cartesian product graphs and Subgraph GNNs in general (detailed in Appendix A.2).

A.1. The Cartesian Product Graph and its Application for `Subgraphormer`

Definition A.1 (Cartesian Product Graph). *Given two graphs G_1 and G_2, their Cartesian product $G_1 \square G_2$ is defined as:*

- *The vertex set $V(G_1 \square G_2) = V(G_1) \times V(G_2)$.*

- *Vertices (u_1, u_2) and (v_1, v_2) in $G_1 \square G_2$ are adjacent if:*
  - *$u_1 = v_1$ and u_2 is adjacent to v_2 in G_2, or*
  - *$u_2 = v_2$ and u_1 is adjacent to v_1 in G_1.*

Thus, we observe the following,

Corollary A.1. *Let G be a given graph, defined by an adjacency A, and assume A doesn't include self loops. The adjacency matrix of the Cartesian product of G with itself; namely, $G \square G$, is given by*

$$\mathcal{A}_{G \square G} \triangleq A \otimes I + I \otimes A, \tag{12}$$

where by $\otimes$ we denote the Kronecker product.

Therefore, we claim,

Proposition 3.2 (Internal and External Connectivities give rise to the Cartesian Product Graph). *The edges induced by the internal and external subgraph connectivities, represented by $\mathcal{A}_G$ and $\mathcal{A}_{G^s}$ (Equations (2) and (3)) represent the connectivity of $\mathcal{A}_{G \square G}$. This implies the relationship:*

$$\mathcal{A}_{G \square G} = \mathcal{A}_{G^s} + \mathcal{A}_G. \tag{6}$$

In particular, we have $\mathcal{A}_{G^s} = A \otimes I, \mathcal{A}_G = I \otimes A$.

The proof is available in Appendix E.

Thus, given Proposition 3.2, our objective is to diagonalize the Laplacian of $G \square G$, which is defined by,

$$\mathcal{L}_{G \square G} = \text{diag}\left(\mathcal{A}_{G \square G} \vec{1}_{n^2}\right) - \mathcal{A}_{G \square G} \tag{13}$$
$$= \text{diag}\left((A \otimes I + I \otimes A)\vec{1}_{n^2}\right) - A \otimes I + I \otimes A,$$

where by $\vec{1}_{n^2}$ we refer to a vector of n^2 ones.

The eigendecomposition of $\mathcal{L}_{G \square G}$, is given in the following proposition,

Proposition 4.1 (Product Graph eigendecomposition). *Consider a graph $G = (A, X)$.[6] The eigenvectors and eigenvalues of the Laplacian matrix of $G \square G$, namely, $\mathcal{L}_{G \square G}$, are $\{(v_i \otimes v_j, \lambda_i + \lambda_j)\}_{i,j=1}^{n^2}$, where $\{(v_i, \lambda_i)\}_{i=1}^{n}$ are the eigenvectors and eigenvalues of the Laplacian matrix of G.*

The proof is available in Appendix E.

We conclude with the following result, stating the complexity of our product graph PE.

Proposition A.1 (Product Graph PE Complexity). *Let $G = (A, X)$ be an undirected graph with n vertices, and consider its Cartesian product graph $G \square G$.*

1. *The time complexity for diagonalizing the Laplacian matrix of $G \square G$ is $\mathcal{O}(n^4)$.*

---

[6]We assume A has no self loops.

2. *For calculating k eigenvectors, where $k \leq n$, the time complexity is $\mathcal{O}(k \cdot n^2)$. This is equivalent to the complexity of computing k eigenvectors for the original graph G.*

The proof is available in Appendix E.

A.2. Establishing the General Relationship Between Cartesian Product Graphs and Subgraph GNNs

In this section we develop the connection between Subgraph GNNs and the Graph Cartesian Product (GCP); the results are summarized in Table 5. We focus specifically on the node-based (Node-Marking) generation policy, over an original graph $G = (A, X)$, and specify the formulation of the following four main connectivities: Internal Subgraph connectivity, External Subgraph connectivity, Global Internal Subgraph connectivity, Global External Subgraph connectivity.

Before delving into the derivation details, we define two graphs, G_s and G_c, corresponding to the "set graph" and the "clique graph", respectively. The "set graph" corresponds to a graph with no edges, thus:

$$A_{G_s} = \vec{0}\vec{0}^T, \tag{14}$$

and the "clique graph" corresponds to a graph where every pair of nodes is connected by an edge, hence:

$$A_{G_c} = \vec{1}\vec{1}^T - I. \tag{15}$$

Recalling the definition of the Cartesian product graph (Definition A.1), we introduce a relevant corollary:

Corollary A.2 (Adjacency of Cartesian Product Graph). *Consider two distinct graphs, $G_1 = (A_1, X_1)$ and $G_2 = (A_2, X_2)$, with n and m vertices, respectively. The adjacency matrix of their Cartesian product, denoted as $G_1 \square G_2$, is given by:*

$$\mathcal{A}_{G_1 \square G_2} = A_{G_1} \otimes I_m + I_n \otimes A_{G_2}, \tag{16}$$

where $\otimes$ denotes the Kronecker product, and I_n and I_m are identity matrices in $\mathbb{R}^{n \times n}$ and $\mathbb{R}^{m \times m}$, respectively.

In this subsection, as our focus is on graphs each containing n nodes, we will omit the subscripts n/m from the identity matrix for simplicity.

Internal Subgraph Connectivity. The adjacency matrix corresponding to the internal subgraph connectivity, $\mathcal{A}_G$, arises from the Cartesian product of the graphs G_s and G, that is,

$$\mathcal{A}_G \triangleq \mathcal{A}_{G_s \square G} = A_{G_s} \otimes I + I \otimes A. \tag{17}$$

By substituting Equation (14), we obtain:

$$\mathcal{A}_{G_s \square G} = I \otimes A. \tag{18}$$

This results is in line with Proposition 3.2.

External Subgraph Connectivity. Similarly, the adjacency matrix for external subgraph connectivity, $\mathcal{A}_{G^s}$, results from the Cartesian product of the graphs G and G_s, that is,

$$\mathcal{A}_{G^s} \triangleq \mathcal{A}_{G \square G_s} = A \otimes I + I \otimes A_{G_s}. \tag{19}$$

After substituting Equation (14), we find:

$$\mathcal{A}_{G \square G_s} = A \otimes I. \tag{20}$$

We note that this is also in line with Proposition 3.2.

As derived in Section 3.2 in the main paper, the connectivity that combines both Internal and External updates can be given by the Cartesian product of G with itself, as given in Equation (5).

Global Internal Subgraph Connectivity. Following the literature (Zhang et al., 2023a; Frasca et al., 2022), assuming no self loops, the adjacency matrix corresponding to the global internal subgraph connectivity is defined as follows:

$$\mathcal{A}_G^{\text{Global}}\Big((s, v), (s', v')\Big) = \delta_{ss'}, \tag{21}$$

where δ denotes the Kronecker delta.

This matrix results from the Cartesian product of G_s and G_c:

$$\mathcal{A}_G^{\text{Global}} \triangleq \mathcal{A}_{G_s \square G_c} = A_{G_s} \otimes I + I \otimes A_{G_c}. \tag{22}$$

Substituting Equations (14) and (15), we obtain:

$$\mathcal{A}_{G_s \square G_c} = \vec{0}\vec{0}^T \otimes I + I \otimes (\vec{1}\vec{1}^T - I) = I \otimes (\vec{1}\vec{1}^T - I). \tag{23}$$

Global External Subgraph Connectivity. Again, assuming no self loops, the adjacency matrix for global external subgraph connectivity is defined by:

$$\mathcal{A}_{G^S}^{\text{Global}}\Big((s, v), (s', v') \Big) = \delta_{vv'}. \tag{24}$$

This matrix can be obtained from the Cartesian product of G_c and G_s:

$$\mathcal{A}_{G^S}^{\text{Global}} \triangleq \mathcal{A}_{G_c \square G_s} = A_{G_c} \otimes I + I \otimes A_{G_s}. \tag{25}$$

Upon substituting Equations (14) and (15), we get:

$$\mathcal{A}_{G_s \square G_c} = (\vec{1}\vec{1}^T - I) \otimes I + I \otimes \vec{0}\vec{0}^T = (\vec{1}\vec{1}^T - I) \otimes I. \tag{26}$$

Analogously to the adjacency that unifies the internal and external aggregations, we also show that the following Cartesian Graph product results in a unifying adjacency for the global internal connectivity and the global external connectivity. This Cartesian product graph is given by $G_c \square G_c$, and therefore, its corresponding adjacency is,

$$\mathcal{A}_{G_c \square G_c} = (\vec{1}\vec{1}^T - I) \otimes I + I \otimes (\vec{1}\vec{1}^T - I). \tag{27}$$

Table 5: Table summarizing the connection between Cartesian Product Graphs and Subgraph GNNs.

Connectivity type	GCP	Adjacency	Visualization	GCP – unified	Adjacency – unified	Visualization – unified
Internal Subgraph Connectivity	$G_s \square G$	$\mathcal{A}_{G_s \square G} \triangleq \mathcal{A}_G = I \otimes A$				
External Subgraph Connectivity	$G \square G_s$	$\mathcal{A}_{G \square G_s} \triangleq \mathcal{A}_{G^S} = A \otimes I$		$G \square G$	$\mathcal{A}_{G \square G} = A \otimes I + I \otimes A$	
Global Internal Subgraph Connectivity	$G_s \square G_c$	$\mathcal{A}_{G_s \square G_c} = I \otimes (\vec{1}\vec{1}^T - I)$				
Global External Subgraph Connectivity	$G_c \square G_s$	$\mathcal{A}_{G_c \square G_s} = (\vec{1}\vec{1}^T - I) \otimes I$		$G_c \square G_c$	$\mathcal{A}_{G_c \square G_c} = (\vec{1}\vec{1}^T - I) \otimes I + I \otimes (\vec{1}\vec{1}^T - I)$	

B. Extending Subgraphormer to any k-tuple

In this section, we extend the formulation of `Subgraphormer`, which was given for 2-tuples of nodes, to any k-tuple. We start in Appendix B.1, by formulating the product graph PE for any k-tuple. This sets the required exposition for Appendix B.2, where we formulate a SAB update to by applied to k-tuples.

B.1. k-tuple Product Graph PE

In this section, we extend the concept of product graph PE to encompass any k-tuple for arbitrary values of k. This generalization stems from an essential observation that arises from the definition of the Cartesian Product Graph (Definition A.1).

Corollary A.2 (Adjacency of Cartesian Product Graph). *Consider two distinct graphs, $G_1 = (A_1, X_1)$ and $G_2 = (A_2, X_2)$, with n and m vertices, respectively. The adjacency matrix of their Cartesian product, denoted as $G_1 \square G_2$, is given by:*

$$\mathcal{A}_{G_1 \square G_2} = A_{G_1} \otimes I_m + I_n \otimes A_{G_2}, \tag{16}$$

where $\otimes$ denotes the Kronecker product, and I_n and I_m are identity matrices in $\mathbb{R}^{n \times n}$ and $\mathbb{R}^{m \times m}$, respectively.

In this paper, our primary focus lies on a specific scenario where the Cartesian Product Graph is applied iteratively to the same graph. For this purpose, we introduce the Cartesian product operator. This operator, when applied to an adjacency matrix A of a graph G, yields the adjacency matrix corresponding to $G^{\square^k}$ - the graph resulting from k-fold recursive Cartesian product of G with itself.

Definition B.1 (Cartesian Product Operator). *The Cartesian product operator is defined as:*

$$\mathcal{C}^k : \mathbb{R}^{n^k \times n^k} \to \mathbb{R}^{n^{k+1} \times n^{k+1}}, \tag{28}$$

applied to an adjacency matrix $A \in \mathbb{R}^{n \times n}$, recursively defined as:

$$\mathcal{C}^k(A) = \mathcal{C}^{k-1}(A) \otimes I_n + I_{n^{k-1}} \otimes A, \tag{29}$$

with the base case being:

$$\mathcal{C}^1(A) = A. \tag{30}$$

It is apparent that for a graph G with adjacency matrix A, applying the operator $\mathcal{C}^2$ to A results in $\mathcal{C}^2(A) = A \otimes I_n + I_n \otimes A$, which is the adjacency matrix corresponding to the Cartesian Product of the graph with itself – Equation (11).

We propose the following proposition regarding the applicability of the Cartesian operator $\mathcal{C}$ across any power k:

Proposition B.1 (Adjacency matrix of $G^{\square^k}$). *For a graph $G = (A, X)$ with adjacency matrix A, the adjacency matrix of $G^{\square^k}$—the graph formed by k-fold Cartesian product of G with itself—is given by:*

$$\mathcal{A}_{G^{\square^k}} = \mathcal{C}^k(A). \tag{31}$$

The proof is given in Appendix E.

Utilizing Proposition B.1, we establish that $\mathcal{C}^k$ effectively constructs the adjacency matrix corresponding to the graph $G^{\square^k}$.

Notably, this operator functions recursively, therefore, to extend positional encoding to any k-tuple, our goal is to formulate a closed-form expression for the adjacency matrix of $G^{\square^k}$; we first introduce the necessary definition.

Definition B.2 ($\mathcal{A}_\mathcal{K}^k$). *Given an adjacency matrix $A \in \mathbb{R}^{n \times n}$ and an index $k \in [\mathcal{K}]$, define $\mathcal{A}_\mathcal{K}^k$ to be the tensor resulting from the Cartesian product of $\mathcal{K}$ matrices, where each matrix is the identity matrix $I_n \in \mathbb{R}^{n \times n}$, except for the k-th factor which is A. That is,*

$$\mathcal{A}_\mathcal{K}^k = I_n \otimes I_n \otimes \cdots \otimes A \otimes \cdots \otimes I_n, \tag{32}$$

where A occupies the k-th position in the Cartesian product.

Thus, we propose the following,

Proposition B.2 (Adjacency matrix of $G^{\square^k}$ – closed form)**.** *Given a graph $G = (A, X)$, with an adjacency matrix A, it holds that,*

$$\mathcal{C}^{\mathcal{K}}(A) = \sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^k. \tag{33}$$

The proof is given in Appendix E.

Having Proposition B.1 and Proposition B.2, we straight forwardly infer the following Corollary,

Corollary B.1. *For a graph $G = (A, X)$, the adjacency matrix of $G^{\square^k}$—the graph formed by k-fold Cartesian product of G with itself—is given by:*

$$\mathcal{A}_{G^{\square^{\mathcal{K}}}} = \sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^k. \tag{34}$$

In what follows, we prove that the matrix $\mathcal{A}_{G^{\square^k}}$ is a valid, binary adjacency matrix.

Proposition B.3 ($\mathcal{A}_{G^{\square^k}}$ – Valid binary adjacency matrix)**.** *Given a graph $G = (A, X)$ with an adjacency matrix A, with no self loops, the matrix given by,*

$$\mathcal{A}_{G^{\square^{\mathcal{K}}}} = \sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^k, \tag{35}$$

is a binary adjacency matrix.

The proof is given in Appendix E.

Having developed this closed form expression for the (binary) adjacency matrix of $G^{\square^k}$, we have the following Proposition, which generalizes Proposition 4.1 to any k-tuple,

Proposition B.4 (k-tuple – Product graph eigendecomposition)**.** *Consider a graph $G = (A, X)$ without self-loops. Define $G^{\square^k}$ as the Cartesian product of G repeated k times, with its adjacency matrix denoted by $\mathcal{A}_{G^{\square^k}}$. The eigenvectors and eigenvalues of the Laplacian matrix for $G^{\square^k}$ can be characterized as follows: for each set of indices $\{i_1, i_2, \ldots, i_k\}$, where $i_j \in \{1, 2, \ldots, n\}$ for each j, there exists an eigenvector-eigenvalue pair given by $(v_{i_1} \otimes v_{i_2} \otimes \ldots \otimes v_{i_k}, \lambda_{i_1} + \lambda_{i_2} + \ldots + \lambda_{i_k})$. Here, $\{(v_i, \lambda_i)\}_{i=1}^n$ represent the eigenvectors and eigenvalues of the Laplacian matrix of the original graph G.*

The proof is given in Appendix E.

The above Proposition enables the computation of positional encodings for any k-tuple, offering significantly more efficiency compared to the conventional method of directly diagonalizing a matrix in $\mathbb{R}^{n^k \times n^k}$.

Theorem B.1 (k-tuple PE efficiency)**.** *Consider a graph $G = (A, X)$ without self-loops. Define $G^{\square^k}$ as the Cartesian product of G repeated $k \geq 2$ times.*

The time complexity of computing the eigendecomposition of the Laplacian matrix of $G^{\square^k}$ is $\mathcal{O}(n^{2k})$.

The proof is given in Appendix E.

We note that diagonalizing directly the Laplacian matrix of $G^{\square^k}$ would instead take $\mathcal{O}(n^{3k})$.

B.2. k-tuple SAB

Building upon the derivation of the adjacency matrix obtained from the Cartesian product of graph G applied recursively k times, as outlined in Equation (34), we can extend the node update mechanisms, initially conceptualized for 2-tuples in Section 4.1, to support k-tuples.

Generalization of Internal/External updates. We recall the 2-tuple case, where the two matrices corresponding the Internal and External aggregations of `Subgraphormer` are given in Equation (5). In our new formalism, using the Cartesian Product Operator, we can write,

$$\mathcal{A}_{G \square G} = \mathcal{A}_2^0 + \mathcal{A}_2^1, \tag{36}$$

where $\mathcal{A}_2^0 = \mathcal{A}_{G^s}$ and $\mathcal{A}_2^1 = \mathcal{A}_G$. Thus the Internal and External updates are given as $\{\mathcal{A}_2^0, \mathcal{A}_2^1\}$.

Table 6: Overview of the graph learning datasets.

Dataset	# Graphs	Avg. # nodes	Avg. # edges	Directed	Prediction task	Metric
ZINC-12K (Sterling and Irwin, 2015)	12,000	23.2	24.9	No	Regression	Mean Abs. Error
ZINC-FULL (Sterling and Irwin, 2015)	249,456	23.2	49.8	No	Regression	Mean Abs. Error
ALCHEMY-12K (Chen et al., 2019)	12,000	10.12	20.9	No	12-task Regression	Mean Abs. Error
OGBG-MOLHIV (Hu et al., 2020)	41,127	25.5	27.5	No	Binary Classification	AUROC
OGBG-MOLBACE (Hu et al., 2020)	1513	34.1	36.9	No	Binary Classification	AUROC
OGBG-MOLESOL (Hu et al., 2020)	1,128	13.3	13.7	No	Regression	Root Mean Squ. Error
PEPTIDES-FUNC (Dwivedi et al., 2022)	15,535	150.9	307.3	No	10-task Classification	Avg. Precision
PEPTIDES-STRUCT (Dwivedi et al., 2022)	15,535	150.9	307.3	No	11-task Regression	Mean Abs. Error

Building on Equation (34), we can extend to a $\mathcal{K}$-tuple scenario where the "Internal/External" updates correspond to a series of adjacencies, denoted by $\{\mathcal{A}^i_{\mathcal{K}}\}^{\mathcal{K}-1}_{i=0}$.

Generalization of Point-wise update. As defined in Section 3, the point-wise update should allow a node v in subgraph s to have access to the root representation, where the roots in the case of a 2-tuple are defined to be, $\{(v,v); v \in V\}$.

While the root nodes in the case of k-tuples can be defined as $\{(\overbrace{v,\ldots,v}^{k \text{ elements}}); v \in V\}$, the correspondence between nodes and subgraphs is not clear.

Consequently, for a k-tuple scenario, we define the point-wise update by mapping the "subgraph" to a subset of $k-1$ nodes within the tuple, and designating the "node" as the remaining element in the tuple. Take, for instance, the tuple $(s_1, s_2, \ldots, s_i, v, s_{i+1}, \ldots, s_k)$: here, the "subgraph" comprises the indices $(s_1, s_2, \ldots, s_i, s_{i+1}, \ldots, s_k)$, while the "node" is represented by v.

Accordingly, we establish k such point-wise updates. To be precise, for each $i \in \{1, 2, \ldots, k\}$, we define the update as follows:

$$\mathcal{A}^i_{\text{point}}\big((v_1, \ldots, v_k), (v'_1, \ldots, v'_k)\big) = \begin{cases} 1 & \text{if } v'_1 = \ldots = v'_k \text{ and } v'_1 = v_j \text{ for all } j \neq i; \\ 0 & \text{otherwise.} \end{cases} \tag{37}$$

This equation defines the function $\mathcal{A}^i_{\text{point}}$, which evaluates the adjacency between two k-tuples of vertices, $(v_1, \ldots, v_k)$ and $(v'_1, \ldots, v'_k)$. The function returns 1, indicating a specific type of adjacency, if the second tuple is a root node – all vertices in the second tuple are identical ($v'_1 = v'_2 = \ldots = v'_k$), and simultaneously, all vertices in the first tuple match this root node for all indices except i, denoted as $v'_1 = v_j$ for all $j \neq i$. In other instances, where these conditions are not satisfied, the function returns 0, signifying the absence of such adjacency.

C. Extended Experimental Section

C.1. Dataset Description

In this section we overview the eight different datasets considered; this is summarized in Table 6.

ZINC-12K and ZINC-FULL Datasets (Sterling and Irwin, 2015; Gómez-Bombarelli et al., 2018; Dwivedi et al., 2023). The ZINC-12K dataset comprises 12,000 molecular graphs, extracted from the ZINC database, which is a collection of commercially available chemical compounds. These molecular graphs vary in size, ranging from 9 to 37 nodes each. In these graphs, nodes correspond to heavy atoms, encompassing 28 distinct atom types. Edges in the graphs represent chemical bonds, with three possible bond types. The primary objective when using this dataset is to perform regression analysis on the constrained solubility (logP) of the molecules. The dataset is pre-partitioned into training, validation, and test sets, containing 10,000, 1,000, and 1,000 molecular graphs, respectively. In its full form, namely, ZINC-FULL, it contains approximately 250,000 molecular graphs. These graphs vary in complexity, with each graph containing again between 9 to 37 nodes, and 16 to 84 edges. The nodes in these graphs also represent heavy atoms, and the dataset includes 28 different types of atoms. The edges, on the other hand, represent bonds between these atoms, and there are 4 distinct types of bonds in the dataset.

Table 7: Test results over the ALCHEMY-12K dataset. Subgraph-based baselines are highlighted in light blue. The top three results are reported as **First**, **Second**, and **Third**.

Model	ALCHEMY-12K (MAE ↓)
GIN (Xu et al., 2018)	0.180 ± 0.006
SignNet (Lim et al., 2022)	**0.113**±0.002
δ-2-GNN (Morris et al., 2020)	0.118 ± 0.001
δ-2-LGNN (Morris et al., 2020)	0.122 ± 0.003
SpeqNet (Morris et al., 2022)	0.115 ± 0.001
GNN-IR (Dupty and Lee, 2022)	0.119 ± 0.002
PF-GNN (Dupty et al., 2021)	**0.111**±0.010
PPGN++(6) (Puny et al., 2023)	**0.109**±0.001
Recon. GNN (Cotta et al., 2021)	0.125 ± 0.001
DS-GNN (NM) (Bevilacqua et al., 2023)	0.116 ± 0.001
GNN-SSWL+ (Zhang et al., 2023a)	0.116 ± 0.002
Subgraphormer	0.114 ± 0.001
Subgraphormer + PE	**0.113**±0.002

OGBG-MOLHIV, OGBG-MOLBACE, OGBG-MOLESOL Datasets (Hu et al., 2020). Those datasets are molecular property prediction datasets, adopted by the Open Graph Benchmark (OGB) from MoleculeNet. These datasets employ a unified featurization for nodes (atoms) and edges (bonds), encapsulating various chemophysical properties.

ALCHEMY-12K Dataset (Chen et al., 2019). The ALCHEMY-12K dataset includes quantum mechanical properties of 12,000 organic molecules containing up to 14 heavy atoms like Carbon (C), Nitrogen (N), Oxygen (O), Fluorine (F), Sulfur (S), and Chlorine (Cl). These properties were computed using the Python-based Simulations of Chemistry Framework (PySCF).

PEPTIDES-FUNC and PEPTIDES-STRUCT Datasets (Dwivedi et al., 2022). The PEPTIDES-FUNC and PEPTIDES-STRUCT datasets consist of atomic graphs representing peptides. In PEPTIDES-FUNC, the task involves multi-label graph classification into ten non-exclusive peptide functional classes. Conversely, PEPTIDES-STRUCT focuses on graph regression to predict eleven three-dimensional structural properties of the peptides.

C.2. Experimental Details

Our experiments were conducted using the PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey and Lenssen, 2019) frameworks, using a single NVIDIA L40 GPU, and for every considered experiment, we show the mean $\pm$ std. of 3 runs with different random seeds. Hyperparameter tuning was performed utilizing the Weight and Biases framework (Biewald, 2020) – see Appendix C.5. All our MLPs feature a single hidden layer equipped with a ReLU non-linearity function. For the encoding of atom numbers and bonds, we utilized learnable embeddings indexed by their respective numbers. An exception was made for the ALCHEMY-12K dataset, where instead we employed a linear layer, as done in Morris et al. (2022). For all our datasets, we used a fixed value of $H = 4$ attention heads.

In the case of the OGBG-MOLHIV, OGBG-MOLESOL, OGBG-MOLBACE, ALCHEMY-12K datasets, we follow Frasca et al. (2022), therefore adding a residual connection between different layers. Additionally, for those datasets, we used linear layers instead of MLPs inside the GIN layers. Moreover, for these four datasets, and for the PEPTIDES datasets, the following pooling mechanism was employed, instead of the one mentioned in Equation (9) (which was used for ZINC-12K and ZINC-FULL),

$$\rho(\mathcal{X}) = \text{MLP}\left(\sum_{s=1}^{n}\left(\frac{1}{n}\sum_{v=1}^{n}\mathcal{X}(s,v)\right)\right). \tag{38}$$

For the PEPTIDES datasets, we also used a residual connection between layers.

C.3. Additional Results

ALCHEMY-12K. The results regarding the ALCHEMY-12K dataset (Chen et al., 2019) is summarized in Table 7, showcasing Subgraphormer outperforming Subgraph GNNs methods.

Table 8: Comparison of the stochastic variant of `Subgraphormer` and `Subgraphormer PE` to DSS-GNN, over the ZINC-12K dataset. Best result for each sampling ratio is in **Bold**.

Model		ZINC-12K (MAE $\downarrow$)
DSS-GNN (EGO+) (Bevilacqua et al., 2022)	100%	0.102 ± 0.003
	50%	0.155 ± 0.007
	20%	0.166 ± 0.005
	5%	0.179 ± 0.001
Subgraphormer	100%	0.067 ± 0.007
	50%	$\mathbf{0.084}\pm0.002$
	20%	0.121 ± 0.007
	5%	0.200 ± 0.017
Subgraphormer + PE	100%	$\mathbf{0.063}\pm0.0003$
	50%	$\mathbf{0.084}\pm0.002$
	20%	$\mathbf{0.120}\pm0.002$
	5%	$\mathbf{0.175}\pm0.006$

Table 9: Comparison of the stochastic variant of `Subgraphormer` and `Subgraphormer PE` to GNN-SSWL+, over the OGBG-MOLBACE dataset. Best result for each sampling ratio is in **Bold**.

Model		OGBG-MOLBACE (ROC-AUC $\uparrow$)
GNN-SSWL+ (Zhang et al., 2023a)	100%	82.70 ± 1.80
	50%	79.99 ± 0.58
	20%	78.04 ± 5.98
	5%	$\mathbf{68.52}\pm5.73$
Subgraphormer	100%	81.62 ± 3.55
	50%	79.49 ± 2.38
	20%	75.27 ± 5.63
	5%	63.05 ± 8.70
Subgraphormer + PE	100%	$\mathbf{84.35}\pm0.65$
	50%	$\mathbf{83.82}\pm2.62$
	20%	$\mathbf{78.77}\pm4.10$
	5%	67.73 ± 5.50

Stochastic Subgraph Sampling. The comparison of our stochastic sampling approach against the proposed method by Bevilacqua et al. (2022) over the ZINC-12K dataset is given in Table 8.

In Tables 9 and 10 we compare our stochastic sampling approach against a stochastic version of GNN-SSWL+. The table demonstrates that `Subgraphormer + PE` consistently surpasses the performance of GNN-SSWL+ across the datasets examined (7 out of 8 combinations of sampling ratios and datasets).

C.3.1. PRODUCT GRAPH PE VS CONCATENATION PE

For completeness, we expand on the other valid choice of subgraph positional encodings, also introduced in Section 4.2 – *concatenation PE*, which is a more general subgraph positional encoding scheme. Specifically, given the Laplacian matrix of G, L, and its eigendecomposition, $L = U^T \Lambda U$, we define the *concatenation PE* for node (s, v) to be the output of an `MLP` acting on the concatenation of $\mathbf{p}_s^{:k} \triangleq [U_{s1}, \ldots, U_{sk}]$ and $\mathbf{p}_v^{:k} \triangleq [U_{v1}, \ldots, U_{vk}]$.

Proposition C.1 (*Concatenation PE* can approximate *product graph PE*)**.** *The* concatenation PE *can approximate (up to an ordering) the* product graph PE *uniformly.*

The proof is given in Appendix E.

Nevertheless, in this paper, we opt for the *product graph PE*, as it demonstrates better performance in general. As an example, we provide a comparison between the *product graph PE* and the *Concatenation PE* over the ZINC-12K dataset in Table 11.

Notably, `Subgraphormer + product graph PE` exhibits superior performance. Interestingly, despite the theoretical capability of `concatenation PE` to implement `product graph PE` (as discussed in Proposition C.1), our empirical findings reveal that `concatenation PE` actually reduces the effectiveness of `Subgraphormer`.

Table 10: Comparison of the stochastic variant of `Subgraphormer` and `Subgraphormer PE` to GNN-SSWL+, over the OGBG-MOLESOL dataset. Best result for each sampling ratio is in **Bold**.

Model		OGBG-MOLESOL (RMSE ↓)
GNN-SSWL+ (Zhang et al., 2023a)	100%	0.837±0.019
	50%	0.886±0.026
	20%	1.180±0.036
	5%	1.299±0.044
Subgraphormer	100%	0.832±0.043
	50%	0.829±0.013
	20%	1.093±0.009
	5%	**1.266**±0.019
Subgraphormer + PE	100%	**0.826**±0.010
	50%	**0.812**±0.001
	20%	**1.041**±0.030
	5%	1.270±0.007

Table 11: A Comparison between different PE variants; including `Subgraphormer` alone. Best results are in **Bold**.

Model	ZINC-12K (MAE ↓)
Subgraphormer	0.067±0.007
Subgraphormer + concatenation PE	0.071±0.003
Subgraphormer + product graph PE	**0.063**±0.0003

C.4. Runtime Comparison

We estimated the training time and inference time of a single epoch (measured in seconds) for our architecture, as well as for three representative baselines: GIN (Xu et al., 2018) (an MPNN), GNN-SSWL+ (Zhang et al., 2023a) (a Subgraph GNN), and GPS (Rampášek et al., 2022) (A Graph Transformer). For a fair comparison, we used an NVIDIA A100 GPU for all methods; the run-time of GPS was taken from their paper (Rampášek et al., 2022). The experiment was performed on the ZINC-12K dataset (using a batch size of 128 for all baselines, except GPS, which used a batch size of 32), and over the OGBG-MOLHIV dataset (using a batch size of 32 for all baselines). results are summarized in the Tables 12 and 13.

As shown in the tables, and as expected, GIN (the MPNN baseline) has the fastest run-time, while exhibiting the lowest performance. Our method and GNN-SSWL+ both outperform GPS in terms of run-time across both datasets. Over the ZINC-12K dataset, our method's run-time is roughly the same when compared to GNN-SSWL+ (our architecture offers a very modest speed advantage). On the OGBG-MOLHIV dataset, GNN-SSWL+ runs slightly faster. Finally, we note that `Subgraphormer` uses fewer parameters than all baselines, on both datasets.

C.5. HyperParameters

In this section, we detail the hyperparameter search conducted for our experiments. We use the same hyperparameter grid for both `Subgraphormer` and `Subgraphormer + PE`. The hyperparameter search configurations for the full bag and stochastic bag settings are presented in Tables 14 and 16, respectively. Additionally, we report the final selected hyperparameters for `Subgraphormer + PE` in both settings in Tables 15 and 17. Notably, in the stochastic bag setting with sampling ratios of 20% and 5% over the ZINC-12K dataset, our model failed to converge, leading us to extend the training to 800 epochs.

Optimizers and Schedulers. For the ZINC-12K and ZINC-FULL datasets, we employ the Adam optimizer paired with a ReduceLROnPlateau scheduler (factor set to 0.5, patience at 20, and a minimum learning rate of 0). A similar setup was used for the ALCHEMY-12K dataset, except the minimum learning rate which was set to 1×10^{-7}. For the OGBG-MOLHIV dataset, we utilized the ASAM optimizer (Kwon et al., 2021) without a scheduler. For both OGBG-MOLESOL and OGBG-MOLBACE, we employed a constant learning rate without any scheduler. Lastly, for the PEPTIDES-FUNC and PEPTIDES-STRUCT datasets, the AdamW optimizer was chosen in conjunction with a cosine annealing scheduler, incorporating 10 warmup epochs.

Table 12: Comparison of training and inference times (and parameter counts) on the ZINC-12K dataset using an NVIDIA A100 GPU. The table presents the time taken (in seconds) to train for one epoch and to perform inference on the test set.

Model	Param.	Train time (s)	Test time (s)	MAE $\downarrow$
GIN (Xu et al., 2018)	500k	1.41 ± 0.22	0.36 ± 0.02	0.163 ± 0.004
GPS (Rampášek et al., 2022)	424k	21 ± N/A	N/A	0.070 ± 0.004
GNN-SSWL+ (Zhang et al., 2023a)	387k	9.65 ± 0.19	1.04 ± 0.03	0.070 ± 0.005
Subgraphormer + PE	293k	9.60 ± 0.10	0.95 ± 0.03	0.063 ± 0.001

Table 13: Comparison of training and inference times (and parameter counts) on the OGBG-MOLHIV dataset using an NVIDIA A100 GPU. The table presents the time taken (in seconds) to train for one epoch and to perform inference on the test set.

Model	Param.	Train time (s)	Test time (s)	ROC-AUC $\downarrow$
GIN (Xu et al., 2018)	1800k	12.65 ± 0.21	1.05 ± 0.08	75.58 ± 1.40
GPS (Rampášek et al., 2022)	558k	96 ± N/A	N/A	78.80 ± 1.01
GNN-SSWL+ (Zhang et al., 2023a)	46k	51.02 ± 0.25	3.073 ± 0.03	79.58 ± 0.35
Subgraphormer + PE	30k	64.62 ± 0.16	3.200 ± 0.1	80.38 ± 1.92

D. Complexity

This section provides an analysis of the computational complexity associated with each aggregation method we presented, as illustrated in Section 3.1. Consider a graph $G = (A, X)$; with notes and edges denoted as V, E, respectively. Our product graph always encompasses $|V|^2$ nodes, and the number of edges is different for each considered aggregation as follows,

1. **Internal:** For $\mathcal{A}_G$, the number of edges is computed by $\sum_{s \in V} \sum_{v \in V} (1 + d(v))$, where $d(v)$ denotes the degree of vertex v (in the original graph G). This simplifies to $|V| \cdot (|V| + |E|) = |V|^2 + |V| \cdot |E|$.

2. **External:** Similarly, for $\mathcal{A}_{G^s}$, the number of edges is given by $\sum_{v \in V} \sum_{s \in V} (1 + d(s))$, which also simplifies to $|V| \cdot (|V| + |E|) = |V|^2 + |V| \cdot |E|$.

3. **Point:** For $\mathcal{A}_{\text{Point}}$, the number of edges is $|V|^2$.

The complexities are summarized in Table 18.

Subgraphormer Complexity. We utilized the GAT convolution for our attention-based aggregations, as introduced by Veličković et al. (2017). Given a graph $G' = (A', X')$, with nodes and edges denoted as V', E', respectively, the time complexity for a single GAT attention head[7] is $\mathcal{O}(|V'| + |E'|)$. Thus, the overall complexity depends on the number of nodes $|V'|$ and the number of edges $|E'|$ in the graph. Specifically, the product graph we process always has $|V|^2$ nodes, while the number of edges are given in Table 18. Thus, SAB exhibits the complexity of $\mathcal{O}(|V|^2 + |V| \cdot |E|)$.

As elaborated in Section 4.2, the computational complexity associated with calculating the product graph PE, which involves k eigenvectors, is expressed as $\mathcal{O}(k \cdot |V|^2)$.

Therefore, the total complexity of Subgraphormer is $\mathcal{O}(k \cdot |V|^2 + |V| \cdot |E|)$.

Run-time in Seconds. The run-time performance was evaluated on the ZINC-12K dataset, where the average run-time, calculated over three different seeds, was 2 hours, 5 minutes, and 24 seconds, with a standard deviation of ±29 seconds. It is important to highlight that the *product graph PE* computation can be executed as a preprocessing step. Notably, this process required less than 10 minutes for the ZINC-12K dataset.

E. Proofs

Proposition 3.1 (GNN-SSWL+ as an MPNN on the product graph). *Consider a graph $G = (A, X)$. Applying a stacking of RGCN layers (Schlichtkrull et al., 2018), interleaved with ReLU activations, on the product graph, as defined via the adjacencies in Equations* (2) *to* (4)*, can implement the GNN-SSWL+ update in Equation* (1)*.*

---

[7]The feature dimension is considered a constant.

Table 14: Hyperparameters search for `Subgraphormer` and `Subgraphormer + PE` in full bag settings.

Dataset	Num. layers	Learning rate	Embedding size	Epochs	Batch size	Dropout	Num. Eigenvectors
ZINC-12K	{6}	{0.001, 0.0005, 0.0003, 0.0001}	{96}	{400}	{128}	{0}	{0, 1, 2, 8, 16}
ZINC-FULL	{6}	{0.001, 0.0005, 0.0003, 0.0001}	{96}	{400}	{128}	{0}	{0, 1, 2, 8, 16}
MOLHIV	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLSOL	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLBACE	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3}	{0, 1, 2, 8, 16}
ALCHEMY-12K	{5}	{0.01, 0.001}	{96}	{400}	{128}	{0}	{0, 1, 2, 8, 16}

Table 15: Best hyperparameters for `Subgraphormer + PE` in full bag settings.

Dataset	Num. layers	Learning rate	Embedding size	Epochs	Batch size	Dropout	Num. Eigenvectors
ZINC-12K	6	0.0005	96	400	128	0	8
ZINC-FULL	6	0.0005	96	400	128	0	16
MOLHIV	2	0.1	60	100	32	0.3	8
MOLSOL	3	0.001	60	100	32	0.5	2
MOLBACE	3	0.001	60	100	32	0.3	16
ALCHEMY-12K	5	0.01	96	400	128	0	16

Proof. We consider an unnormalized variant of RGCN, defined as:

$$\mathcal{X}^{t+1} = \text{RGCN}^t\left(\mathcal{X}^t, \{\mathcal{A}_i\}_{i=1}^M\right), \tag{39}$$

$$\text{RGCN}^t\left(\mathcal{X}^t, \{\mathcal{A}_i\}_{i=1}^M\right) = \mathcal{X}^t\mathbf{W}_0^t + \sum_{i=1}^M \mathcal{A}_i\mathcal{X}^t\mathbf{W}_i^t. \tag{40}$$

We assume the input features at the first layer, $\mathcal{X}^t$, are provided as 1-hot vectors, and that the parameterized function f^t, which is applied in Equation (1) also outputs 1-hot vectors.

We aim to show that any given layer t of a GNN-SSWL+, as per Equation (1), can be implemented by this RGCN model. The proof involves the following steps:

1. Encode uniquely each of the inputs (individually) to the parameterized function f^t in Equation (1),

$$\mathcal{X}(s,v)^t, \mathcal{X}(v,v)^t, \{\mathcal{X}(s,v')^t\}_{v'\sim_G v}, \{\mathcal{X}(s',v)^t\}_{s'\sim_G s}. \tag{41}$$

2. Encode uniquely the input of f^t as a whole,

$$\left(\mathcal{X}(s,v)^t, \mathcal{X}(v,v)^t, \{\mathcal{X}(s,v')^t\}_{v'\sim_G v}, \{\mathcal{X}(s',v)^t\}_{s'\sim_G s}\right). \tag{42}$$

3. Implement the parameterized function f^t.

4. Ensure the output features are 1-hot features – to maintain generality across layers.

Step 1: The unique encoding of the inputs can be achieved as follows:

$$\mathcal{X}(s,v)^t \to \mathcal{X}(s,v)^t, \tag{43}$$

$$\mathcal{X}(v,v)^t \to \left(\mathcal{A}_{\text{point}}\mathcal{X}^t\right)(s,v), \tag{44}$$

$$\{\mathcal{X}(s,v')^t\}_{v'\sim_G v} \to \left(\mathcal{A}_G\mathcal{X}^t\right)(s,v), \tag{45}$$

$$\{\mathcal{X}(s',v)^t\}_{s'\sim_G s} \to \left(\mathcal{A}_{G^s}\mathcal{X}^t\right)(s,v), \tag{46}$$

For Equations (45) and (46), the right-hand sides effectively count the number of each element in the set, since the input features are 1-hot vectors. For Equations (43) and (44), the right hand side is actually equal to the left hand side.

Table 16: Hyperparameters search for `Subgraphormer` and `Subgraphormer + PE` in stochastic sampling settings.

Dataset	Sampling Ratio	Num. layers	Learning rate	Embedding size	Epochs	Batch size	Dropout	Num. Eigenvectors
ZINC-12K	50%	{6}	{0.001, 0.0005, 0.0003, 0.0001}	{96}	{400}	{128}	{0}	{0, 1, 2, 8, 16}
MOLHIV	50%	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLSOL	50%	{3, 6, 12}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLBACE	50%	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3}	{0, 1, 2, 8, 16}
PEPTIDES-FUNC	30%	{5}	{0.005, 0.003, 0.001}	{96}	{200}	{128}	{0}	{0, 1, 2, 8, 16}
PEPTIDES-STRUCT	30%	{4}	{0.005, 0.003, 0.001}	{96}	{200}	{128}	{0}	{0, 1, 2, 8, 16}
ZINC-12K	20%	{6}	{0.001, 0.0005, 0.0003, 0.0001}	{96}	{400, 800}	{128}	{0}	{0, 1, 2, 8, 16}
MOLHIV	20%	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLSOL	20%	{3, 6, 12}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLBACE	20%	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3}	{0, 1, 2, 8, 16}
ZINC-12K	5%	{6}	{0.001, 0.0005, 0.0003, 0.0001}	{96}	{400, 800}	{128}	{0}	{0, 1, 2, 8, 16}
MOLHIV	5%	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLSOL	5%	{3, 6, 12}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3, 0.5}	{0, 1, 2, 8, 16}
MOLBACE	5%	{2, 3}	{0.1, 0.01, 0.001}	{60}	{100}	{32}	{0.3}	{0, 1, 2, 8, 16}

Table 17: Best hyperparameters for `Subgraphormer + PE` in stochastic sampling settings.

Dataset	Sampling Ratio	Num. layers	Learning rate	Embedding size	Epochs	Batch size	Dropout	Num. Eigenvectors
ZINC-12K	50%	6	0.0005	96	400	128	0	8
MOLHIV	50%	2	0.1	60	100	32	0.3	8
MOLSOL	50%	3	0.001	60	100	32	0.5	2
MOLBACE	50%	3	0.001	60	100	32	0.3	8
PEPTIDES-FUNC	30%	5	0.003	96	200	128	0	16
PEPTIDES-STRUCT	30%	4	0.005	96	200	128	0	16
ZINC-12K	20%	6	0.0005	96	800	128	0	8
MOLHIV	20%	2	0.1	60	100	32	0.3	8
MOLSOL	20%	12	0.001	60	100	32	0.5	1
MOLBACE	20%	3	0.001	60	100	32	0.3	8
ZINC-12K	5%	6	0.0005	96	800	128	0	8
MOLHIV	5%	2	0.1	60	100	32	0.3	8
MOLSOL	5%	12	0.001	60	100	32	0.5	2
MOLBACE	5%	3	0.001	60	100	32	0.3	2

Step 2: Assuming that $\mathcal{X} \in \mathbb{R}^{n^2 \times d}$, the RGCN that uniquely encodes the input of f^t as a whole is defined as follows,

$$
\begin{aligned}
&\text{RGCN}^t\left(\mathcal{X}^t, \{\mathcal{A}_G, \mathcal{A}_{G^s}, \mathcal{A}_{\text{point}}\}\right) \\
&= \mathcal{X}^t \mathbf{W}_0^t + \mathcal{A}_{\text{point}} \mathcal{X}^t \mathbf{W}_{\text{point}}^t + \mathcal{A}_G \mathcal{X}^t \mathbf{W}_G^t + \mathcal{A}_{G^s} \mathcal{X}^t \mathbf{W}_{G^s}^t,
\end{aligned} \tag{47}
$$

where the weight matrices are defined as:

$$
\mathbf{W}_0^t = \begin{pmatrix} I_{d\times d} & 0_{d\times d} & 0_{d\times d} & 0_{d\times d} \end{pmatrix}, \tag{48}
$$

$$
\mathbf{W}_{\text{point}}^t = \begin{pmatrix} 0_{d\times d} & I_{d\times d} & 0_{d\times d} & 0_{d\times d} \end{pmatrix}, \tag{49}
$$

$$
\mathbf{W}_G^t = \begin{pmatrix} 0_{d\times d} & 0_{d\times d} & I_{d\times d} & 0_{d\times d} \end{pmatrix}, \tag{50}
$$

$$
\mathbf{W}_{G^s}^t = \begin{pmatrix} 0_{d\times d} & 0_{d\times d} & 0_{d\times d} & I_{d\times d} \end{pmatrix}, \tag{51}
$$

More specifically, it holds that,

$$
\mathcal{X}^t \mathbf{W}_0^t = \begin{pmatrix} \mathcal{X}^t & 0_{d\times d} & 0_{d\times d} & 0_{d\times d} \end{pmatrix}, \tag{52}
$$

$$
\mathcal{A}_{\text{point}} \mathcal{X}^t \mathbf{W}_{\text{point}}^t = \begin{pmatrix} 0_{d\times d} & \mathcal{A}_{\text{point}} \mathcal{X}^t & 0_{d\times d} & 0_{d\times d\cdot} \end{pmatrix}, \tag{53}
$$

$$
\mathcal{A}_G \mathcal{X}^t \mathbf{W}_G^t = \begin{pmatrix} 0_{d\times d} & 0_{d\times d} & \mathcal{A}_G \mathcal{X}^t & 0_{d\times d} \end{pmatrix}, \tag{54}
$$

$$
\mathcal{A}_{G^s} \mathcal{X}^t \mathbf{W}_{G^s}^t = \begin{pmatrix} 0_{d\times d} & 0_{d\times d} & 0_{d\times d} & \mathcal{A}_{G^s} \mathcal{X}^t \end{pmatrix}. \tag{55}
$$

Table 18: Complexity analysis of our aggregations.

Aggregation Type	Number of edges
$\mathcal{A}_G$	$\mathcal{O}(|V|^2 + |V| \cdot |E|)$
$\mathcal{A}_{G^S}$	$\mathcal{O}(|V|^2 + |V| \cdot |E|)$
$\mathcal{A}_{\text{Point}}$	$\mathcal{O}(|V|^2)$

Thus, since RGCN is the sum of those, it holds that,

$$\text{RGCN}^t\Big(\mathcal{X}^t, \{\mathcal{A}_G, \mathcal{A}_{G^S}, \mathcal{A}_{\text{point}}\}\Big)$$
$$= \begin{pmatrix} \mathcal{X}^t & \mathcal{A}_{\text{point}}\mathcal{X}^t & \mathcal{A}_G\mathcal{X}^t & \mathcal{A}_{G^S}\mathcal{X}^t \end{pmatrix}.$$

This concatenation is a unique encoding of the input for f^t.

Step 3: To prove this step, we start with the following theorem from Yun et al. (2019) about the memorization power of ReLU networks:

Theorem E.1. *Consider a dataset $\{x_i, y_i\}_{i=1}^N \in \mathbb{R}^d \times \mathbb{R}^{d_y}$, with each x_i being distinct and every $y_i \in \{0, 1\}^{d_y}$, for all i. There exists a 4-layer fully connected ReLU neural network $f_\theta : \mathbb{R}^d \to \mathbb{R}^{d_y}$ that perfectly maps each x_i to its corresponding y_i, i.e., $f_\theta(x_i) = y_i$ for all i.*

Employing this theorem, we define the x_is as all possible (distinct) rows derived from the output of Step 2, represented as:

$$\begin{pmatrix} \mathcal{X}^t & \mathcal{A}_{\text{point}}\mathcal{X}^t & \mathcal{A}_G\mathcal{X}^t & \mathcal{A}_{G^S}\mathcal{X}^t \end{pmatrix},$$

with each corresponding y_i being the output of the parameterized function f^t over the original input that is represented by this x_i. Given the finite nature of our graph set, the input set is also finite, i.e., N is finite.

Hence, in light of Theorem E.1, a 4-layer fully connected ReLU neural network capable of implementing f^t exists. This network can be equivalently realized using four layers of RGCN. Specifically, by setting $\{\mathbf{W}_i^t\}_{i=1^M}$ to zero and utilizing only $\mathbf{W}_0^t$ for $t \in \{0, 1, 2, 3\}$, we effectively mimic the operation of this 4-layer fully connected ReLU network.

Step 4: Finally, employing this same logic, we can again use Theorem E.1 and map those outputs of step 3 back to 1-hot vectors. $\square$

Proposition 3.2 (Internal and External Connectivities give rise to the Cartesian Product Graph). *The edges induced by the internal and external subgraph connectivities, represented by $\mathcal{A}_G$ and $\mathcal{A}_{G^S}$ (Equations (2) and (3)) represent the connectivity of $\mathcal{A}_{G\square G}$. This implies the relationship:*

$$\mathcal{A}_{G\square G} = \mathcal{A}_{G^S} + \mathcal{A}_G. \tag{6}$$

In particular, we have $\mathcal{A}_{G^S} = A \otimes I, \mathcal{A}_G = I \otimes A$.

Proof. Without loss of generality, consider the matrix $(I \otimes A)$ indexed at $\big((s, v), (s', v')\big)$, we have,

$$(I \otimes A)\big((s, v), (s', v')\big) \triangleq I(s, s') \cdot A(v, v'). \tag{56}$$

In this context, edges exist if and only if $s = s'$ and v, v' are neighbors in the original graph G. This is in direct correspondence with the definition of $\mathcal{A}_G$, as outlined in Equation (2). The proof for $A \otimes I = \mathcal{A}_{G^S}$ follows a similar line of reasoning. $\square$

Proposition 4.1 (Product Graph eigendecomposition). *Consider a graph $G = (A, X)$.[8] The eigenvectors and eigenvalues of the Laplacian matrix of $G\square G$, namely, $\mathcal{L}_{G\square G}$, are $\{(v_i \otimes v_j, \lambda_i + \lambda_j)\}_{i,j=1}^{n^2}$, where $\{(v_i, \lambda_i)\}_{i=1}^n$ are the eigenvectors and eigenvalues of the Laplacian matrix of G.*

---

[8]We assume A has no self loops.

Proof. By definition,

$$\mathcal{L}_{G\square G} = \mathcal{D}_{G\square G} - \mathcal{A}_{G\square G}, \tag{57}$$

where $\mathcal{D}_{G\square G}$ is defined as

$$\mathcal{D}_{G\square G} \triangleq \text{diag}(\mathcal{A}_{G\square G}\vec{1}_{n^2}), \tag{58}$$

and recalling Equation (12),

$$\mathcal{D}_{G\square G} = \text{diag}\left((A\otimes I + I\otimes A)\vec{1}_{n^2}\right). \tag{59}$$

Using tensor product rules, we have,

$$
\begin{aligned}
\mathcal{D}_{G\square G} &\triangleq \text{diag}\left(A\otimes I + I\otimes A)\vec{1}_{n^2}\right) \\
&= \text{diag}\left(A\otimes I + I\otimes A)(\vec{1}_n\otimes\vec{1}_n)\right) \\
&= \text{diag}\left(A\vec{1}_n\otimes I\vec{1}_n + I\vec{1}_n\otimes A\vec{1}_n\right) \\
&= D\otimes I + I\otimes D,
\end{aligned}
\tag{60}
$$

where $D = A\vec{1}_n$.

Substituting Equation (60) and Equation (12) in Equation (57) we have,

$$\mathcal{L}_{G\square G} = D\otimes I + I\otimes D - A\otimes I + I\otimes A. \tag{61}$$

Therefore,

$$
\begin{aligned}
\mathcal{L}_{G\square G} &= D - A\otimes I + I\otimes D - A \\
&= L\otimes I + I\otimes L.
\end{aligned}
\tag{62}
$$

Thus, the eigenspace of $\mathcal{L}_{G\square G}$ is given by $\{(v_i\otimes v_j, \lambda_i + \lambda_j)\}_{i,j=1}^{n^2}$. $\quad\square$

Proposition A.1 (Product Graph PE Complexity). *Let $G = (A, X)$ be an undirected graph with n vertices, and consider its Cartesian product graph $G\square G$.*

1. *The time complexity for diagonalizing the Laplacian matrix of $G\square G$ is $\mathcal{O}(n^4)$.*

2. *For calculating k eigenvectors, where $k \leq n$, the time complexity is $\mathcal{O}(k \cdot n^2)$. This is equivalent to the complexity of computing k eigenvectors for the original graph G.*

Proof. The proof stems directly from the arguments established in Proposition 4.1, which states that the computation of the eigendecomposition of $\mathcal{L}_{G\square G}$ primarily involves diagonalizing the Laplacian matrix of the original graph G, which incurs a computational cost of $\mathcal{O}(n^3)$. Additionally, to obtain the complete set of n^2 eigenvectors for the graph $G\square G$, an extra $\mathcal{O}(n^4)$ operations are necessary, resulting in a total complexity of $\mathcal{O}(n^4)$.

It is important to note that when the objective is to compute only k eigenvectors, where $k \leq n$, the computational complexity is reduced to $\mathcal{O}(k \cdot n^2)$, since in this case we only need to obtain k eigenvectors. This is congruent with the complexity involved in calculating the same number of eigenvectors for the original graph G, which is also $\mathcal{O}(k \cdot n^2)$, for example by applying the Lanczos Algorithm (Lanczos, 1950) or the Implicitly Restarted Arnoldi Methods (Lehoucq et al., 1998; Lee et al., 2009), as done by widely used libraries such as Scipy (Virtanen et al., 2020). Therefore, our approach offers a computational advantage in scenarios where a subset of eigenvectors ($k \leq n$) suffices; which consist of most cases in practice. $\quad\square$

Proposition B.1 (Adjacency matrix of $G^{\square^k}$). *For a graph $G = (A, X)$ with adjacency matrix A, the adjacency matrix of $G^{\square^k}$—the graph formed by k-fold Cartesian product of G with itself—is given by:*

$$\mathcal{A}_{G^{\square^k}} = \mathcal{C}^k(A). \tag{31}$$

Proof. The proof proceeds by induction.

Base Case: For $k = 2$, we must show that $\mathcal{A}_{G^{\square 2}} \triangleq \mathcal{C}^2(A)$.

From Corollary A.1, we have:

$$\mathcal{A}_{G^{\square 2}} \triangleq \mathcal{A}_{G \square G} = A \otimes I_n + I_n \otimes A. \tag{63}$$

Using Equation (29) and Equation (30), we get:

$$\mathcal{C}^2(A) = \mathcal{C}^1(A) \otimes I_n + I_n \otimes A = A \otimes I_n + I_n \otimes A. \tag{64}$$

Thus, from Equation (63) and Equation (64) we obtain,

$$\mathcal{A}_{G^{\square 2}} = \mathcal{C}^2(A), \tag{65}$$

this verifies the base case.

Inductive Assumption: Assume the proposition holds for some integer k, i.e.,

$$\mathcal{A}_{G^{\square k}} \triangleq \mathcal{C}^k(A), \tag{66}$$

then we must demonstrate its validity for $k + 1$:

$$\mathcal{A}_{G^{\square k+1}} \triangleq \mathcal{C}^{k+1}(A). \tag{67}$$

Inductive Step: Let $G_1 = G^{\square k}$ and $G_2 = G$. Given that G has n nodes, G_1 has n^k nodes, and G_2 has n nodes. Recalling Equation (16), we deduce:

$$\mathcal{A}_{G^{\square k+1}} = \mathcal{A}_{G^{\square k} \square G} = \mathcal{A}_{G_1 \square G_2} = \mathcal{A}_{G^{\square k}} \otimes I_n + I_{n^k} \otimes A, \tag{68}$$

where $\mathcal{A}_{G^{\square k}}$ is the adjacency of G_1 and A is the adjacency of G_2.

Applying the inductive assumption from Equation (66) to Equation (68), we obtain:

$$\mathcal{A}_{G_1 \square G_2} = \mathcal{C}^k(A) \otimes I_n + I_{n^k} \otimes A. \tag{69}$$

The right-hand side of Equation (69) corresponds exactly to the definition of $\mathcal{C}^{k+1}$ as specified in Equation (29).

Hence, we conclude that:

$$\mathcal{A}_{G^{\square k+1}} = \mathcal{A}_{G^{\square k} \square G} = \mathcal{A}_{G_1 \square G_2} = \mathcal{C}^{k+1}(A). \tag{70}$$

This concludes the proof. $\square$

Proposition B.2 (Adjacency matrix of $G^{\square k}$ – closed form). *Given a graph $G = (A, X)$, with an adjacency matrix A, it holds that,*

$$\mathcal{C}^{\mathcal{K}}(A) = \sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^k. \tag{33}$$

Proof. We will prove by induction.

Base Case: The base case, for $\mathcal{K} = 2$, is obtained straight-forwardly from the definition of $\mathcal{A}_{\mathcal{K}}^k$, recalling Equation (32). Formally, by the definition of $\mathcal{C}^k$ – Definition B.1, we have,

$$\mathcal{C}^2(A) = A \otimes I_n + I_n \otimes A. \tag{71}$$

We note that,

$$\mathcal{A}_2^0 = A \otimes I_n, \tag{72}$$
$$\mathcal{A}_2^1 = I_n \otimes A, \tag{73}$$

thus, it holds that,

$$
\begin{aligned}
\mathcal{C}^2(A) &= A \otimes I_n + I_n \otimes A \\
&= \mathcal{A}_2^0 + \mathcal{A}_2^1 \\
&= \sum_{k=0}^{1} \mathcal{A}_2^k,
\end{aligned}
\tag{74}
$$

which completes the base case.

Inductive Assumption: Assume the proposition holds for some integer $\mathcal{K}$, i.e.,

$$
\mathcal{C}^{\mathcal{K}}(A) = \sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^k,
\tag{75}
$$

then we must demonstrate its validity for $\mathcal{K}+1$:

$$
\mathcal{C}^{\mathcal{K}+1}(A) = \sum_{k=0}^{\mathcal{K}} \mathcal{A}_{\mathcal{K}+1}^k.
\tag{76}
$$

Inductive Step: By the definition of the Cartesian operator – Definition B.1, it holds that,

$$
\mathcal{C}^{\mathcal{K}+1}(A) = \mathcal{C}^{\mathcal{K}}(A) \otimes I_n + I_{n^{\mathcal{K}}} \otimes A.
\tag{77}
$$

Substituting Equation (75) to Equation (77) we obtain,

$$
\mathcal{C}^{\mathcal{K}+1}(A) = \Big(\sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^k \Big) \otimes I_n + I_{n^{\mathcal{K}}} \otimes A.
\tag{78}
$$

Since the tensor product operator, $\otimes$, is linear, and recalling the definition of Equation (32) it holds that,

$$
\begin{aligned}
\mathcal{C}^{\mathcal{K}+1}(A) &= \Big(\sum_{k=0}^{\mathcal{K}-1} \overbrace{\mathcal{A}_{\mathcal{K}}^k \otimes I_n}^{\triangleq \mathcal{A}_{\mathcal{K}+1}^k} \Big) + \overbrace{\triangleq I_{n^{\mathcal{K}}} \otimes A}^{\triangleq \mathcal{A}_{\mathcal{K}+1}^{\mathcal{K}}} \\
&= \sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}+1}^k + \mathcal{A}_{\mathcal{K}+1}^{\mathcal{K}} \\
&= \sum_{k=0}^{\mathcal{K}} \mathcal{A}_{\mathcal{K}+1}^k.
\end{aligned}
\tag{79}
$$

This completes the proof. $\qquad\square$

Proposition B.3 ($\mathcal{A}_{G^{\Box k}}$ – Valid binary adjacency matrix). *Given a graph $G = (A, X)$ with an adjacency matrix A, with no self loops, the matrix given by,*

$$
\mathcal{A}_{G^{\Box \mathcal{K}}} = \sum_{k=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^k,
\tag{35}
$$

is a binary adjacency matrix.

Proof. Since for any k the matrix given by $\mathcal{A}_{\mathcal{K}}^k$ is binary, it is sufficient to show that given k, k', such that $k \neq k'$, it holds that,

$$
\langle \mathcal{A}_{\mathcal{K}}^k, \mathcal{A}_{\mathcal{K}}^{k'} \rangle_F = 0,
\tag{80}
$$

where by $\langle \cdot, \cdot \rangle_F$ we denote the Frobenius inner product.

Recalling Definition B.2, we have,

$$
\begin{aligned}
\langle \mathcal{A}_{\mathcal{K}}^{k}, \mathcal{A}_{\mathcal{K}}^{k'} \rangle_F &= \langle I \otimes I \otimes \cdots \otimes A \otimes \cdots \otimes I, I \otimes \cdots \otimes A \otimes \cdots \otimes I \otimes I \rangle_F \\
&= \mathrm{Tr}\Big((I \otimes I \otimes \cdots \otimes A \otimes \cdots \otimes I) \cdot (I \otimes \cdots \otimes A \otimes \cdots \otimes I \otimes I)^T \Big) \\
&= \mathrm{Tr}\Big((I \otimes I \otimes \cdots \otimes A \otimes \cdots \otimes I) \cdot (I^T \otimes \cdots \otimes A^T \otimes \cdots \otimes I^T \otimes I^T) \Big)
\end{aligned}
$$

Using the the fact that $k \neq k'$, we obtain that,

$$
\begin{aligned}
\langle \mathcal{A}_{\mathcal{K}}^{k}, \mathcal{A}_{\mathcal{K}}^{k'} \rangle_F &= \mathrm{Tr}\Big(II^T \otimes \cdots \otimes II^T \otimes \cdots \otimes II^T \otimes AI^T \otimes II^T \otimes \cdots \otimes II^T \otimes IA^T \otimes II^T \otimes \cdots \otimes II^T \Big) \\
&= \mathrm{Tr}(II^T) \cdot \mathrm{Tr}(II^T) \cdot \ldots \cdot \mathrm{Tr}(AI^T) \cdot \mathrm{Tr}(II^T) \cdot \ldots \cdot \mathrm{Tr}(II^T) \cdot \mathrm{Tr}(IA^T) \cdot \mathrm{Tr}(II^T) \cdot \ldots \cdot \mathrm{Tr}(II^T).
\end{aligned}
$$

Since we assume no self loops in the original graph G, we know that the diagonal of A and A^T is zero, therefore, the Trace of both AI^T and IA^T is zero. Thus, for any $k \neq k'$, it holds that,

$$
\langle \mathcal{A}_{\mathcal{K}}^{k}, \mathcal{A}_{\mathcal{K}}^{k'} \rangle_F = 0. \tag{81}
$$

$\square$

Proposition B.4 (k-tuple – Product graph eigendecomposition)**.** *Consider a graph $G = (A, X)$ without self-loops. Define $G^{\square^k}$ as the Cartesian product of G repeated k times, with its adjacency matrix denoted by $\mathcal{A}_{G^{\square^k}}$. The eigenvectors and eigenvalues of the Laplacian matrix for $G^{\square^k}$ can be characterized as follows: for each set of indices $\{i_1, i_2, \ldots, i_k\}$, where $i_j \in \{1, 2, \ldots, n\}$ for each j, there exists an eigenvector-eigenvalue pair given by $(v_{i_1} \otimes v_{i_2} \otimes \ldots \otimes v_{i_k}, \lambda_{i_1} + \lambda_{i_2} + \ldots + \lambda_{i_k})$. Here, $\{(v_i, \lambda_i)\}_{i=1}^{n}$ represent the eigenvectors and eigenvalues of the Laplacian matrix of the original graph G.*

Proof. We are looking for the eigenvectors of $L_{G^{\square^k}}$, defined as

$$
L_{G^{\square^k}} = D_{G^{\square^k}} - \mathcal{A}_{G^{\square^k}}, \tag{82}
$$

where $D_{G^{\square^k}}$ can be written as

$$
\begin{aligned}
D_{G^{\square^k}} &\triangleq \mathrm{diag}\left(\mathcal{A}_{G^{\square^k}} \vec{1}_{n^k} \right) \\
&= \mathrm{diag}\left(\sum_{k'=0}^{\mathcal{K}-1} \mathcal{A}_{\mathcal{K}}^{k'} \vec{1}_{n^k} \right) \\
&= \sum_{k'=0}^{\mathcal{K}-1} \mathrm{diag}\left(\mathcal{A}_{\mathcal{K}}^{k'} \vec{1}_{n^k} \right), \tag{83}
\end{aligned}
$$

where the last equality is obtained from the linearity of the diag operator.

Recalling Definition B.2, for a given k', it holds that,

$$
\begin{aligned}
\mathcal{A}_{\mathcal{K}}^{k'} \vec{1}_{n^k} &= (I_n \otimes I_n \otimes \ldots \otimes I_n \otimes A \otimes I_n \otimes \ldots \otimes I_n) \cdot (\vec{1}_n \otimes \vec{1}_n \otimes \ldots \otimes \vec{1}_n) \\
&= I_n \vec{1}_n \otimes I_n \vec{1}_n \otimes \ldots I_n \vec{1}_n \otimes A \vec{1}_n \otimes I_n \vec{1}_n \otimes \ldots I_n \vec{1}_n \\
&= \vec{1}_n \otimes \vec{1}_n \otimes \ldots \vec{1}_n \otimes A \vec{1}_n \otimes \vec{1}_n \otimes \ldots \otimes \vec{1}_n.
\end{aligned}
$$

Since the degree matrix D of the original graph G satisfies $D = \mathrm{diag}\, A \vec{1}_n$ we obtain,

$$
\mathrm{diag}\left(\mathcal{A}_{\mathcal{K}}^{k'} \vec{1}_{n^k} \right) = I_n \otimes I_n \otimes \ldots \otimes I_n \otimes D \otimes I_n \otimes I_n \otimes \ldots \otimes I_n. \tag{84}
$$

For simplicity, we will extend Definition B.2 to any calligraphic matrix; i.e., we define,

$$
\mathcal{D}_{\mathcal{K}}^{k'} \triangleq I_n \otimes I_n \otimes \ldots \otimes I_n \otimes D \otimes I_n \otimes I_n \otimes \ldots \otimes I_n, \tag{85}
$$

such that the matrix D occupies the k'-th slot.

Substituting Equation (85) and Equation (84) to Equation (83), we obtain,

$$D_{G^{\Box k}} = \sum_{k'=0}^{\mathcal{K}-1} \mathcal{D}_{\mathcal{K}}^{k'}. \tag{86}$$

Plugging Equation (86) and Equation (34) to Equation (82),

$$L_{G^{\Box k}} = \sum_{k'=0}^{\mathcal{K}-1} \left(\mathcal{D}_{\mathcal{K}}^{k'} - \mathcal{A}_{\mathcal{K}}^{k'} \right). \tag{87}$$

For a given k', recalling Definition B.2 and Equation (85) it holds that,

$$
\begin{aligned}
\mathcal{D}_{\mathcal{K}}^{k'} - \mathcal{A}_{\mathcal{K}}^{k'} &\triangleq I_n \otimes I_n \otimes \ldots \otimes I_n \otimes D \otimes I_n \otimes I_n \otimes \ldots \otimes I_n - I_n \otimes I_n \otimes \ldots \otimes I_n \otimes A \otimes I_n \otimes I_n \otimes \ldots \otimes I_n \\
&= I_n \otimes I_n \otimes \ldots \otimes I_n \otimes L \otimes I_n \otimes I_n \otimes \ldots \otimes I_n \\
&\triangleq \mathcal{L}_{\mathcal{K}}^{k'}.
\end{aligned}
\tag{88}
$$

Thus,

$$L_{G^{\Box k}} = \sum_{k'=0}^{\mathcal{K}-1} \mathcal{L}_{\mathcal{K}}^{k'}, \tag{89}$$

which states that the eigenvector-eigenvalue pairs are given by $(v_{i_1} \otimes v_{i_2} \otimes \ldots \otimes v_{i_k}, \lambda_{i_1} + \lambda_{i_2} + \ldots + \lambda_{i_k})$, for any $i_j \in [n]$, where $j \in [k]$; given that $\{(v_i, \lambda_i)\}_{i=1}^n$ represents the eigenvectors and eigenvalues of the Laplacian matrix of the original graph G. $\qquad \square$

Theorem B.1 (k-tuple PE efficiency). *Consider a graph $G = (A, X)$ without self-loops. Define $G^{\Box k}$ as the Cartesian product of G repeated $k \geq 2$ times.*

The time complexity of computing the eigendecomposition of the Laplacian matrix of $G^{\Box k}$ is $\mathcal{O}(n^{2k})$.

Proof. We begin by diagonalizing the Laplacian matrix of the graph G. This process requires $\mathcal{O}(n^3)$ time complexity.

Following this, we utilize the results in Proposition B.4 to determine the eigenvectors and eigenvalues of the Cartesian power of the graph, denoted as $G^{\Box k}$. The computation of these eigenvectors and their corresponding eigenvalues requires $\mathcal{O}(n^{2k})$ operations.

Therefore, the overall computational complexity for this procedure is $\mathcal{O}(n^{2k} + n^3)$. However, since $k \geq 2$ the dominating term is $\mathcal{O}(n^{2k})$. Thus, we can conclude that the total complexity is $\mathcal{O}(n^{2k})$. $\qquad \square$

Proposition C.1 (*Concatenation PE* can approximate *product graph PE*). *The* concatenation PE *can approximate (up to an ordering) the* product graph PE *uniformly.*

Proof. The full embedding of a node v in the graph is defined as

$$\mathbf{p}_v \triangleq [U_{v,0}, \ldots, U_{v,n-1}]. \tag{90}$$

The function $F^{\text{prod}} : \mathbb{R}^{2n} \to \mathbb{R}^{n^2}$ is defined as follows:

$$F^{\text{prod}}(\mathbf{p}_s \parallel \mathbf{p}_v)_i = \texttt{flatten}(\mathbf{p}_s \cdot \mathbf{p}_v^T). \tag{91}$$

This function is defined over the compact set $[-1, 1]^{2n}$, and maps to the compact set $[-1, 1]^{n^2}$. Thus is due to the fact that the entries of $\mathbf{p}_{v'}$ for any node v' are components of normalized eigenvectors.

Therefore, the function F^{prod} is continuous and defined over a compact set, implying that it can be approximated by a `MLP` via the Universal Approximation Theorem (Hornik, 1991; Cybenko, 1989) acting on the input $(\mathbf{p}_s \parallel \mathbf{p}_v)$. This completes the proof. $\qquad \square$
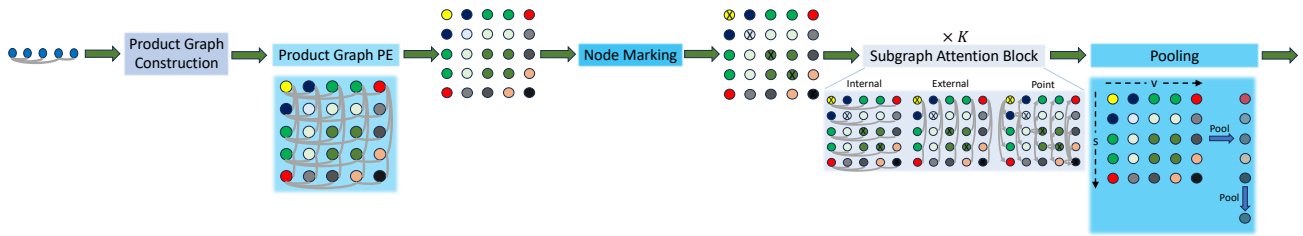
Figure 3: A deep overview of `Subgraphormer`. Given an input graph, the process begins with the construction of the product graph. This is followed by the computation of product graph PE, illustrated through varying colors on the nodes. Node-Marking is then implemented, depicted by black exes (✗) on diagonal nodes. The process continues with the application of K Subgraph Attention Blocks (SABs), characterized by three distinct connectivities: Internal, External, and Point. The final stage involves the pooling layer, which initially aggregates data across the node dimension to form subgraph representations, and subsequently across the subgraph dimension.

F. `Subgraphormer` Figure

We present a detailed figure illustrating the architecture of our `Subgraphormer` model – Figure 3. We begin with the construction of the Product Graph. Subsequently, we apply product graph PE and node-marking, as detailed in Section 4.2. This step is followed by stacking of K Subgraph Attention Blocks (see Section 4.1). Finally, the process concludes with the integration of a pooling layer that returns a graph representation.