Structure-Aware Transformer for Graph Representation Learning

Dexiong Chen [*1 2] **Leslie O'Bray** [*1 2] **Karsten Borgwardt** [1 2]

Abstract

The Transformer architecture has gained growing attention in graph representation learning recently, as it naturally overcomes several limitations of graph neural networks (GNNs) by avoiding their strict structural inductive biases and instead only encoding the graph structure via positional encoding. Here, we show that the node representations generated by the Transformer with positional encoding do not necessarily capture structural similarity between them. To address this issue, we propose the Structure-Aware Transformer, a class of simple and flexible graph Transformers built upon a new self-attention mechanism. This new self-attention incorporates structural information into the original self-attention by extracting a subgraph representation rooted at each node before computing the attention. We propose several methods for automatically generating the subgraph representation and show theoretically that the resulting representations are at least as expressive as the subgraph representations. Empirically, our method achieves state-of-the-art performance on five graph prediction benchmarks. Our structure-aware framework can leverage any existing GNN to extract the subgraph representation, and we show that it systematically improves performance relative to the base GNN model, successfully combining the advantages of GNNs and Transformers. Our code is available at `https://github.com/BorgwardtLab/SAT`.

1. Introduction

Graph neural networks (GNNs) have been established as powerful and flexible tools for graph representation learning, with successful applications in drug discovery (Gaudelet et al., 2021), protein design (Ingraham et al., 2019), social network analysis (Fan et al., 2019), and so on. A large class of GNNs build multilayer models, where each layer operates on the previous layer to generate new representations using a message-passing mechanism (Gilmer et al., 2017) to aggregate local neighborhood information.

While many different message-passing strategies have been proposed, some critical limitations have been uncovered in this class of GNNs. These include the limited expressiveness of GNNs (Xu et al., 2019; Morris et al., 2019), as well as known problems such as over-smoothing (Li et al., 2018; 2019; Chen et al., 2020; Oono & Suzuki, 2020) and over-squashing (Alon & Yahav, 2021). Over-smoothing manifests as all node representations converging to a constant after sufficiently many layers, while over-squashing occurs when messages from distant nodes are not effectively propagated through certain "bottlenecks" in a graph, since too many messages get compressed into a single fixed-length vector. Designing new architectures beyond neighborhood aggregation is thus essential to solve these problems.

Transformers (Vaswani et al., 2017), which have proved to be successful in natural language understanding (Vaswani et al., 2017), computer vision (Dosovitskiy et al., 2020), and biological sequence modeling (Rives et al., 2021), offer the potential to address these issues. Rather than only aggregating local neighborhood information in the message-passing mechanism, the Transformer architecture is able to capture interaction information between any node pair via a single self-attention layer. Moreover, in contrast to GNNs, the Transformer avoids introducing any structural inductive bias at intermediate layers, addressing the expressivity limitation of GNNs. Instead, it encodes structural or positional information about nodes only into input node features, albeit limiting how much information it can learn from the graph structure. Integrating information about the graph structure into the Transformer architecture has thus gained growing attention in the graph representation learning field. However, most existing approaches only encode positional relationships between nodes, rather than explicitly encoding the structural relationships. As a result, they may not identify structural similarities between nodes and could fail to model the *structural interaction* between nodes (see Figure 1). This could explain why their performance

[*]Equal contribution [1]Department of Biosystems Science and Engineering, ETH Zürich, Switzerland [2]SIB Swiss Institute of Bioinformatics, Switzerland. Correspondence to: Dexiong Chen <dexiong.chen@bsse.ethz.ch>, Leslie O'Bray <leslie.obray@bsse.ethz.ch>.

Proceedings of the 39th International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022. Copyright 2022 by the author(s).

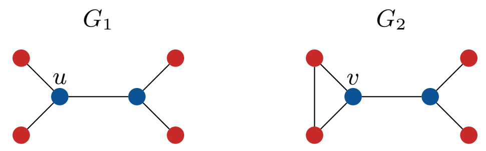


Figure 1: *Position-aware* vs. *structure-aware*: Using a positional encoding based on shortest paths in G_1 and G_2 respectively (assuming all edges have equal weight), node u and v would receive identical encodings since their shortest paths to all other nodes are the same in both graphs. However, their structures are different, with v forming a triangle with its red neighbors.

was dominated by sparse GNNs in several tasks (Dwivedi et al., 2022).

Contribution In this work, we address the critical question of how to encode structural information into a Transformer architecture. Our principal contribution is to introduce a flexible *structure-aware* self-attention mechanism that explicitly considers the graph structure and thus captures structural interaction between nodes. The resulting class of Transformers, which we call the Structure-Aware Transformer (SAT), can provide structure-aware representations of graphs, in contrast to most existing position-aware Transformers for graph-structured data. Specifically:

- We reformulate the self-attention mechanism in Vaswani et al. (2017) as a kernel smoother and extend the original exponential kernel on node features to also account for local structures, by extracting a subgraph representation centered around each node.
- We propose several methods for automatically generating the subgraph representations, enabling the resulting kernel smoother to simultaneously capture structural and attributed similarities between nodes. The resulting representations are theoretically guaranteed to be at least as expressive as the subgraph representations.
- We demonstrate the effectiveness of SAT models on five graph and node property prediction benchmarks by showing it achieves better performance than state-of-the-art GNNs and Transformers. Furthermore, we show how SAT can easily leverage any GNN to compute the node representations which incorporate subgraph information and outperform the base GNN, making it an effortless enhancer of any existing GNN.
- Finally, we show that we can attribute the performance gains to the structure-aware aspect of our architecture, and showcase how SAT is more interpretable than the classic Transformer with an absolute encoding.

We will present the related work and relevant background in Sections 2 and 3 before presenting our method in Section 4 and our experimental findings in Section 5.

2. Related Work

We present here the work most related to ours, namely the work stemming from message passing GNNs, positional representations on graphs, and graph Transformers.

Message passing graph neural networks Message passing graph neural networks have recently been one of the leading methods for graph representation learning. An early seminal example is the GCN (Kipf & Welling, 2017), which was based on performing convolutions on the graph. Gilmer et al. (2017) reformulated the early GNNs into a framework of message passing GNNs, which has since then become the predominant framework of GNNs in use today, with extensive examples (Hamilton et al., 2017; Xu et al., 2019; Corso et al., 2020; Hu et al., 2020b; Veličković et al., 2018; Li et al., 2020a; Yang et al., 2022). However, as mentioned above, they suffer from problems of limited expressiveness, over-smoothing, and over-squashing.

Absolute encoding Because of the limited expressiveness of GNNs, there has been some recent research into the use of *absolute encoding* (Shaw et al., 2018), which consists of adding or concatenating positional or structural representations to the input node features. While it is often called an *absolute positional encoding*, we refer to it more generally as an *absolute encoding* to include both positional and structural encoding, which are both important in graph modeling. Absolute encoding primarily considers position or location relationships between nodes. Examples of position-based methods include the Laplacian positional encoding (Dwivedi & Bresson, 2021; Kreuzer et al., 2021), Weisfeiler–Lehman-based positional encoding (Zhang et al., 2020), and random walk positional encoding (RWPE) (Li et al., 2020b; Dwivedi et al., 2022), while distance-based methods include distances to a predefined set of nodes (You et al., 2019) and shortest path distances between pairs of nodes (Zhang et al., 2020; Li et al., 2020b). Dwivedi et al. (2022) extend these ideas by using a trainable absolute encoding.

Graph Transformers While the absolute encoding methods listed above can be used with message passing GNNs, they also play a crucial role in the (graph) Transformer architecture. Graph Transformer (Dwivedi & Bresson, 2021) provided an early example of how to generalize the Transformer architecture to graphs, using Laplacian eigenvectors as an absolute encoding and computing attention on the immediate neighborhood of each node, rather than on the full graph. SAN (Kreuzer et al., 2021) also used the Laplacian eigenvectors for computing an absolute encoding, but computed attention on the full graph, while distinguishing between true and created edges. Many graph Transformer methods also use a *relative encoding* (Shaw et al., 2018) in addition to absolute encoding. This strategy incorporates representations of the relative position or distances between nodes on the graph directly into the self-attention mechanism, as opposed to the absolute encoding which is only applied once to the input node features. Mialon et al. (2021) propose a relative encoding by means of kernels on graphs to bias the self-attention calculation, which is then able to incorporate positional information into Transformers via the choice of kernel function. Other recent work seeks to incorporate structural information into the graph Transformer, for example by encoding some carefully selected graph theoretic properties such as centrality measures and shortest path distances as positional representations (Ying et al., 2021) or by using GNNs to integrate the graph structure (Rong et al., 2020; Jain et al., 2021; Mialon et al., 2021; Shi et al., 2021).

In this work, we combine the best of both worlds from message passing GNNs and from the Transformer architecture. We incorporate both an absolute as well as a novel relative encoding that explicitly incorporates the graph structure, thereby designing a Transformer architecture that takes both local and global information into account.

3. Background

In the following, we refer to a graph as $G = (V, E, \mathbf{X})$, where the node attributes for node $u \in V$ is denoted by $x_u \in \mathcal{X} \subset \mathbb{R}^d$ and the node attributes for all nodes are stored in $\mathbf{X} \in \mathbb{R}^{n\times d}$ for a graph with n nodes.

3.1. Transformers on Graphs

While GNNs use the graph structure explicitly, Transformers remove that explicit structure, and instead infer relations between nodes by leveraging the node attributes. In this sense, the Transformer (Vaswani et al., 2017) ignores the graph structure and rather considers the graph as a (multi-) set of nodes, and uses the self-attention mechanism to infer the similarity between nodes. The Transformer itself is composed of two main blocks: a self-attention module followed by a feed-forward neural network. In the self-attention module, the input node features $\mathbf{X}$ are first projected to query ($\mathbf{Q}$), key ($\mathbf{K}$) and value ($\mathbf{V}$) matrices through a linear projection such that $\mathbf{Q} = \mathbf{X}\mathbf{W}_\mathbf{Q}$, $\mathbf{K} = \mathbf{X}\mathbf{W}_\mathbf{K}$ and $\mathbf{V} = \mathbf{X}\mathbf{W}_\mathbf{V}$ respectively. We can compute the self-attention via

$$\mathrm{Attn}(\mathbf{X}) := \mathrm{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_{out}}})\mathbf{V} \in \mathbb{R}^{n\times d_{out}}, \quad (1)$$

where d_{out} refers to the dimension of $\mathbf{Q}$, and $\mathbf{W}_\mathbf{Q}, \mathbf{W}_\mathbf{K}, \mathbf{W}_\mathbf{V}$ are trainable parameters. It is common to use *multi-head* attention, which concatenates multiple instances of Eq. (1) and has shown to be effective in practice (Vaswani et al., 2017). Then, the output of the self-attention is followed by a skip-connection and a feed-forward network (FFN), which jointly compose a Transformer layer, as shown below:

$$\begin{aligned}\mathbf{X}' &= \mathbf{X} + \mathrm{Attn}(\mathbf{X}),\\ \mathbf{X}'' &= \mathrm{FFN}(\mathbf{X}') := \mathrm{ReLU}(\mathbf{X}'W_1)W_2.\end{aligned} \quad (2)$$

Multiple layers can be stacked to form a Transformer model, which ultimately provides node-level representations of the graph. As the self-attention is equivariant to permutations of the input nodes, the Transformer will always generate the same representations for nodes with the same attributes regardless of their locations and surrounding structures in the graph. It is thus necessary to incorporate such information into the Transformer, generally via absolute encoding.

Absolute encoding Absolute encoding refers to adding or concatenating the positional or structural representations of the graph to the input node features before the main Transformer model, such as the Laplacian positional encoding (Dwivedi & Bresson, 2021) or RWPE (Dwivedi et al., 2022). The main shortcoming of these encoding methods is that they generally do not provide a measure of the structural similarity between nodes and their neighborhoods.

Self-attention as kernel smoothing As noticed by Mialon et al. (2021), the self-attention in Eq. (1) can be rewritten as a kernel smoother

$$\mathrm{Attn}(x_v) = \sum_{u\in V} \frac{\kappa_{\exp}(x_v, x_u)}{\sum_{w\in V}\kappa_{\exp}(x_v, x_w)} f(x_u), \ \forall v \in V, \quad (3)$$

where $f(x) = \mathbf{W}_\mathbf{V} x$ is the linear value function and $\kappa_{\exp}$ is a (non-symmetric) exponential kernel on $\mathbb{R}^d \times \mathbb{R}^d$ parameterized by $\mathbf{W}_\mathbf{Q}$ and $\mathbf{W}_\mathbf{K}$:

$$\kappa_{\exp}(x, x') := \exp\left(\langle \mathbf{W}_\mathbf{Q} x, \mathbf{W}_\mathbf{K} x'\rangle / \sqrt{d_{out}}\right), \quad (4)$$

where $\langle\cdot,\cdot\rangle$ is the dot product on $\mathbb{R}^d$. With this form, Mialon et al. (2021) propose a relative positional encoding strategy via the product of this kernel and a diffusion kernel on the graph, which consequently captures the positional similarity between nodes. However, this method is only position-aware, in contrast to our structure-aware encoding that will be presented in Section 4.

4. Structure-Aware Transformer

In this section, we will describe how to encode the graph structure into the self-attention mechanism and provide a class of Transformer models based on this framework.

4.1. Structure-Aware Self-Attention

As presented above, self-attention in the Transformer can be rewritten as a kernel smoother where the kernel is a trainable exponential kernel defined on node features, and which only

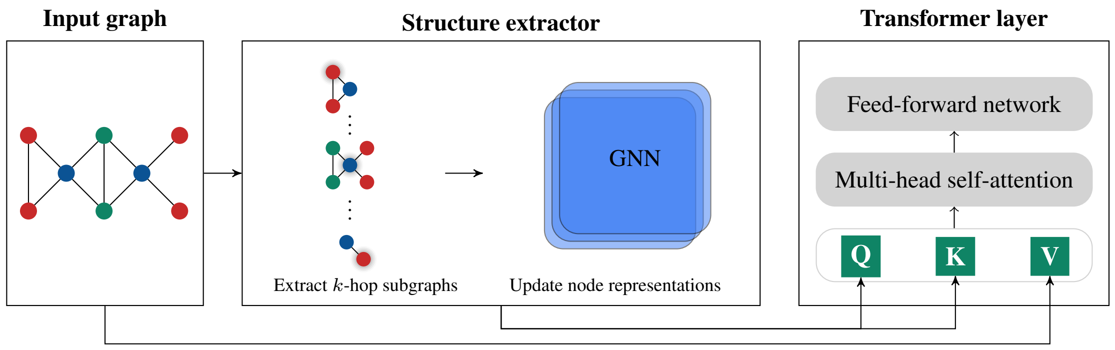

Input graph
Structure extractor
Transformer layer
Extract k-hop subgraphs
GNN
Update node representations
Feed-forward network
Multi-head self-attention
Q
K
V

Figure 2: Overview of an example SAT layer that uses the k-subgraph GNN extractor as its structure extractor. The structure extractor generates structure-aware node representations which are used to compute the query ($\mathbf{Q}$) and key ($\mathbf{K}$) matrices in the Transformer layer. Structure-aware node representations are generated in the k-subgraph GNN extractor by first extracting the k-hop subgraph centered at each node (here, $k = 1$) and then using a GNN on each subgraph to generate a node representations using the full subgraph information. While the structure extractor can use any class of subgraphs, the one illustrated here defined on the class of k-hop subgraphs has a reasonable computation-expressiveness trade-off.

captures attributed similarity between a pair of nodes. The problem with this kernel smoother is that it cannot filter out nodes that are structurally different from the node of interest when they have the same or similar node features. In order to also incorporate the structural similarity between nodes, we consider a more generalized kernel that additionally accounts for the local substructures around each node. By introducing a set of subgraphs centered at each node, we define our structure-aware attention as:

$$\text{SA-attn}(v) := \sum_{u \in V} \frac{\kappa_{\text{graph}}(S_G(v), S_G(u))}{\sum_{w \in V} \kappa_{\text{graph}}(S_G(v), S_G(w))} f(x_u), \tag{5}$$

where $S_G(v)$ denotes a subgraph in G centered at a node v associated with node features $\mathbf{X}$ and κ_{graph} can be any kernel that compares a pair of subgraphs. This new self-attention function not only takes the attributed similarity into account but also the structural similarity between subgraphs. It thus generates more expressive node representations than the original self-attention, as we will show in Section 4.4. Moreover, this self-attention is no longer equivariant to any permutation of nodes but only to nodes whose features and subgraphs coincide, which is a desirable property.

In the rest of the paper, we will consider the following form of κ_{graph} that already includes a large class of expressive and computationally tractable models:

$$\kappa_{\text{graph}}(S_G(v), S_G(u)) = \kappa_{\exp}(\varphi(v, G), \varphi(u, G)), \tag{6}$$

where $\varphi(u, G)$ is a *structure extractor* that extracts vector representations of some subgraph centered at u with node features $\mathbf{X}$. We provide several alternatives of the structure extractor below. It is worth noting that our structure-aware self-attention is flexible enough to be combined with any model that generates representations of subgraphs, including GNNs and (differentiable) graph kernels. For notational simplicity, we assume there are no edge attributes, but our method can easily incorporate edge attributes as long as the structure extractor can accommodate them. The edge attributes are consequently not considered in the self-attention computation, but are incorporated into the structure-aware node representations. In the structure extractors presented in this paper, this means that edge attributes were included whenever the base GNN was able to handle edge attributes.

k-subtree GNN extractor A straightforward way to extract local structural information at node u is to apply any existing GNN model to the input graph with node features $\mathbf{X}$ and take the output node representation at u as the subgraph representation at u. More formally, if we denote by $\text{GNN}_G^{(k)}$ an arbitrary GNN model with k layers applied to G with node features $\mathbf{X}$, then

$$\varphi(u, G) = \text{GNN}_G^{(k)}(u). \tag{7}$$

This extractor is able to represent the k-subtree structure rooted at u (Xu et al., 2019). While this class of structure extractors is fast to compute and can flexibly leverage any existing GNN, they cannot be more expressive than the Weisfeiler–Lehman test due to the expressiveness limitation of message passing GNNs (Xu et al., 2019). In practice, a small value of k already leads to good performance, while not suffering from over-smoothing or over-squashing.

k-subgraph GNN extractor A more expressive extractor is to use a GNN to directly compute the representation of the entire k-hop subgraph centered at u rather than just the node representation u. Recent work has explored the idea of using subgraphs rather than subtrees around a node in GNNs, with positive experimental results (Zhang & Li, 2021; Wijesinghe & Wang, 2022), as well as being strictly more powerful than the 1-WL test (Zhang & Li, 2021). We follow the same setup as is done in Zhang & Li (2021), and adapt our GNN extractor to utilize the entire k-hop subgraph. The k-subgraph GNN extractor aggregates the updated node representations of all nodes within the k-hop neighborhood using a pooling function such as summation. Formally, if we denote by $\mathcal{N}_k(u)$ the k-hop neighborhood of node u including itself, the representation of a node u is:

$$\varphi(u, G) = \sum_{v \in \mathcal{N}_k(u)} \mathrm{GNN}_G^{(k)}(v). \tag{8}$$

We observe that prior to the pooling function, the k-subgraph GNN extractor is equivalent to using the k-subtree GNN extractor within each k-hop subgraph. So as to capture the attributed similarity as well as structural similarity, we augment the node representation from k-subgraph GNN extractor with the original node features via concatenation. While this extractor provides more expressive subgraph representations than the k-subtree extractor, it requires enumerating all k-hop subgraphs, and consequently does not scale as well as the k-subtree extractor to large datasets.

Other structure extractors Finally, we present a list of other potential structure extractors for different purposes. One possible choice is to directly learn a number of "hidden graphs" as the "anchor subgraphs" to represent subgraphs for better model interpretability, by using the concepts introduced in Nikolentzos & Vazirgiannis (2020). While Nikolentzos & Vazirgiannis (2020) obtain a vector representation of the input graph by counting the number of matching walks between the whole graph and each of the hidden graphs, one could extend this to the node level by comparing the hidden graphs to the k-hop subgraph centered around each node. The adjacency matrix of the hidden graphs is a trainable parameter in the network, thereby enabling end-to-end training to identify which subgraph structures are predictive. Then, for a trained model, visualizing the learned hidden graphs provides useful insights about the structural motifs in the dataset.

Furthermore, more domain-specific GNNs could also be used to extract potentially more expressive subgraph representations. For instance, Bodnar et al. (2021) recently proposed a new kind of message passing scheme operating on regular cell complexes which benefits from provably stronger expressivity for molecules. Our self-attention mechanism can fully benefit from the development of more domain-specific and expressive GNNs.

Finally, another possible structure extractor is to use a non-parametric graph kernel (*e.g.* a Weisfeiler-Lehman graph kernel) on the k-hop subgraphs centered around each node. This provides a flexible way to combine graph kernels and deep learning, which might offer new theoretical insights into the link between the self-attention and kernel methods.

4.2. Structure-Aware Transformer

Having defined our structure-aware self-attention function, the other components of the Structure-Aware Transformer follow the Transformer architecture as described in Section 3.1; see Figure 2 for a visual overview. Specifically, the self-attention function is followed by a skip-connection, a FFN and two normalization layers before and after the FFN. In addition, we also include the degree factor in the skip-connection, which was found useful for reducing the overwhelming influence of highly connected graph components (Mialon et al., 2021), i.e.,

$$x_v' = x_v + 1/\sqrt{d_v}\, \text{SA-attn}(v), \tag{9}$$

where d_v denotes the degree of node v. After a Transformer layer, we obtain a new graph with the same structure but different node features $G' = (V, E, \mathbf{X}')$, where $\mathbf{X}'$ corresponds to the output of the Transformer layer.

Finally, for graph property prediction, there are various ways to aggregate node-level representations into a graph representation, such as by taking the average or sum. Alternatively, one can use the embedding of a virtual [CLS] node (Jain et al., 2021) that is attached to the input graph without any connectivity to other nodes. We compare these approaches in Section 5.

4.3. Combination with Absolute Encoding

While the self-attention in Eq. (5) is structure-aware, most absolute encoding techniques are only position-aware and could therefore provide complementary information. Indeed, we find that the combination leads to further performance improvements, which we show in Section 5. We choose to use the RWPE (Dwivedi et al., 2022), though any other absolute positional representations, including learnable ones, can also be used.

We further argue that only using absolute positional encoding with the Transformer would exhibit a too relaxed structural inductive bias which is not guaranteed to generate similar node representations even if two nodes have similar local structures. This is due to the fact that distance or Laplacian-based positional representations generally serve as structural or positional signatures but do not provide a measure of structural similarity between nodes, especially in the inductive case where two nodes are from different graphs. This is also empirically affirmed in Section 5 by their relatively worse performance without using our structural encoding. In contrast, the subgraph representations used in the structure-aware attention can be tailored to measure the structural similarity between nodes, and thus generate similar node-level representations if they possess similar attributes and surrounding structures. We can formally state this in the following theorem:

Theorem 1. *Assume that f is a Lipschitz mapping with the Lipschitz constant denoted by Lip(f) and the structure extractor φ is bounded by a constant C_φ on the space of subgraphs. For any pair of nodes v and v' in two graphs $G = (V, E, \mathbf{X})$ and $G' = (V', E', \mathbf{X}')$ with the same number of nodes $|V| = |V'|$, the distance between their representations after the structure-aware attention is bounded by:*

$$\begin{aligned} \|\textit{SA-attn}(v) - \textit{SA-attn}(v')\| \leq C_1[\|h_v - h'_{v'}\| \\ + D(\mathbf{H}, \mathbf{H}')] + C_2 D(\mathbf{X}, \mathbf{X}'), \end{aligned} \tag{10}$$

where $C_1, C_2 > 0$ are constants depending on $|V|$, Lip(f), C_φ and spectral norms of the parameters in SA-attn, whose expressions are given in the Appendix, and $h_w := \varphi(w, G)$ denotes the subgraph representation at node w for any $w \in V$ and $h'_{w'} := \varphi(w', G')$ similarly, and $\mathbf{H} = (h_w)_{w \in V}$ and $\mathbf{H}' = (h'_{w'})_{w' \in V'}$ denote the multiset of subgraph representations in G and G' respectively. Denoting by $\Pi(V, V')$ the set of permutations from V to V', D is an optimal matching metric between two multisets of representations with the same cardinality, defined as

$$D(\mathbf{X}, \mathbf{X}') := \inf_{\pi \in \Pi(V, V')} \sup_{w \in V} \|x_w - x'_{\pi(w)}\|.$$

The proof is provided in the Appendix. The metric D is an optimal matching metric between two multisets which measures how different they are. This theorem shows that two node representations from the SA-attn are similar if the graphs that they belong to have similar multisets of node features and subgraph representations overall, and at the same time, the subgraph representations at these two nodes are similar. In particular, if two nodes belong to the same graph, *i.e.* $G = G'$, then the second and last terms on the right side of Eq. (10) are equal to zero and the distance between their representations is thus constrained by the distance between their corresponding subgraph representations. However, for Transformers with absolute positional encoding, the distance between two node representations is not constrained by their structural similarity, as the distance between two positional representations does not necessarily characterize how structurally similar two nodes are. Despite stronger inductive biases, we will show that our model is still sufficiently expressive in the next section.

4.4. Expressivity Analysis

The expressive power of graph Transformers compared to classic GNNs has hardly been studied, since the soft structural inductive bias introduced in absolute encoding is generally hard to characterize. Thanks to the unique design of our SAT, which relies on a subgraph structure extractor, it becomes possible to study the expressiveness of the output representations. More specifically, we formally show that the node representation from a structure-aware attention layer is at least as expressive as its subgraph representation given by the structure extractor, following the injectivity of the attention function with respect to the query:

Table 1: Comparison of SAT to SOTA methods on graph regression and classification tasks. ZINC results use edge weights where applicable, otherwise without edge weights. * indicates we obtained the results ourselves by adapting the code provided by the original paper. ↑ means that higher is better for the performance metric; ↓ indicates lower is better.

	ZINC ↓	**CLUSTER ↑**	**PATTERN ↑**
# GRAPHS	12,000	12,000	14,000
AVG. # NODES	23.2	117.2	118.9
AVG. # EDGES	49.8	4,303.9	6,098.9
METRIC	MAE	ACCURACY	ACCURACY
GIN	0.387±0.015	64.716±1.553	85.590±0.011
GAT	0.384±0.007	70.587±0.447	78.271±0.186
PNA	0.188±0.004	67.077±0.977*	86.567±0.075
TRANSFORMER+RWPE	0.310±0.005	29.622±0.176	86.183±0.019
GRAPH TRANSFORMER	0.226±0.014	73.169±0.622	84.808±0.068
SAN	0.139±0.006	76.691±0.650	86.581±0.037
GRAPHORMER	0.122±0.006	–	–
K-SUBTREE SAT	0.102±0.005	77.751±0.121	**86.865±0.043**
K-SUBGRAPH SAT	**0.094±0.008**	**77.856±0.104**	86.848±0.037

Theorem 2. *Assume that the space of node attributes $\mathcal{X}$ is countable. For any pair of nodes v and v' in two graphs $G = (V, E, \mathbf{X})$ and $G' = (V', E', \mathbf{X}')$, assume that there exist a node u_1 in V such that $x_{u_1} \neq x_w$ for any $w \in V$ and a node u_2 in V such that its subgraph representation $\varphi(u_2, G) \neq \varphi(w, G)$ for any $w \in V$. Then, there exists a set of parameters and a mapping $f : \mathcal{X} \to \mathbb{R}^{d_{out}}$ such that their representations after the structure-aware attention are different, i.e. SA-attn$(v) \neq$ SA-attn(v'), if their subgraph representations are different, i.e. $\varphi(v, G) \neq \varphi(v', G')$.*

Note that the assumptions made in the theorem are mild as one can always add some absolute encoding or random noise to make the attributes of one node different from all other nodes, and similarly for subgraph representations. The countable assumption on $\mathcal{X}$ is generally adopted for expressivity analysis of GNNs (*e.g.* Xu et al. (2019)). We assume f to be any mapping rather than just a linear function as in the definition of the self-attention function since it can be practically approximated by a FFN in multi-layer Transformers through the universal approximation theorem (Hornik, 1991). Theorem 2 suggests that if the structure extractor is sufficiently expressive, the resulting SAT model can also be at least equally expressive. Furthermore, more expressive extractors could lead to more expressively powerful SAT models and thus better prediction performance, which is also empirically confirmed in Section 5.

Table 2: Comparison of SAT to SOTA methods on OGB datasets.

	OGBG-PPA ↑	OGBG-CODE2 ↑
# GRAPHS	158,100	452,741
AVG. # NODES	243.4	125.2
AVG. # EDGES	2,266.1	124.2
METRIC	ACCURACY	F1 SCORE
GCN	0.6839±0.0084	0.1507±0.0018
GCN-VIRTUAL NODE	0.6857±0.0061	0.1595±0.0018
GIN	0.6892±0.0100	0.1495±0.0023
GIN-VIRTUAL NODE	0.7037±0.0107	0.1581±0.0026
DEEPERGCN	0.7712±0.0071	–
EXPC	**0.7976±0.0072**	–
TRANSFORMER	0.6454±0.0033	0.1670±0.0015
GRAPHTRANS	–	0.1830±0.0024
K-SUBTREE SAT	0.7522±0.0056	**0.1937±0.0028**

5. Experiments

In this section, we evaluate SAT models versus several SOTA methods for graph representation learning, including GNNs and Transformers, on five graph and node prediction tasks, as well as analyze the different components of our architecture to identify what drives the performance. In summary, we discovered the following aspects about SAT:

- The structure-aware framework achieves SOTA performance on graph and node classification tasks, outperforming SOTA graph Transformers and sparse GNNs.
- Both instances of the SAT, namely k-subtree and k-subgraph SAT, *always* improve upon the base GNN it is built upon, highlighting the improved expressiveness of our structure-aware approach.
- We show that incorporating the structure via our structure-aware attention brings a notable improvement relative to the vanilla Transformer with RWPE that just uses node attribute similarity instead of also incorporating structural similarity. We also show that a small value of k already leads to good performance, while not suffering from over-smoothing or over-squashing.
- We show that choosing a proper absolute positional encoding and a readout method improves performance, but to a much lesser extent than incorporating the structure into the approach.

Furthermore, we note that SAT achieves SOTA performance while only considering a small hyperparameter search space. Performance could likely be further improved with more hyperparameter tuning.

5.1. Datasets and Experimental Setup

We assess the performance of our method with five medium to large benchmark datasets for node and graph property prediction, including ZINC (Dwivedi et al., 2020), CLUSTER (Dwivedi et al., 2020), PATTERN (Dwivedi et al., 2020), OGBG-PPA (Hu et al., 2020a) and OGBG-CODE2 (Hu et al., 2020a).

We compare our method to the following GNNs: GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), GIN (Xu et al., 2019), PNA (Corso et al., 2020), DeeperGCN (Li et al., 2020a), and ExpC (Yang et al., 2022). Our comparison partners also include several recently proposed Transformers on graphs, including the original Transformer with RWPE (Dwivedi et al., 2022), Graph Transformer (Dwivedi & Bresson, 2021), SAN (Kreuzer et al., 2021), Graphormer (Ying et al., 2021) and GraphTrans (Jain et al., 2021), a model that uses the vanilla Transformer on top of a GNN.

All results for the comparison methods are either taken from the original paper or from Dwivedi et al. (2020) if not available. We consider k-subtree and k-subgraph SAT equipped with different GNN extractors, including GCN, GIN, GraphSAGE and PNA. For OGBG-PPA and OGBG-CODE2, we do not run experiments for k-subgraph SAT models due to large memory requirements. Full details on the datasets, experimental setup, and hyperparameters are provided in the Appendix.

5.2. Comparison to State-of-the-Art Methods

We show the performance of SATs compared to other GNNs and Transformers in Table 1 and 2. SAT models consistently outperform SOTA methods on these datasets, showing its ability to combine the benefits of both GNNs and Transformers. In particular, for the CODE2 dataset, our SAT models outperform SOTA methods by a large margin despite a relatively small number of parameters and minimal hyperparameter tuning, which will put it at the first place on the OGB leaderboard.

5.3. SAT Models vs. Sparse GNNs

Table 3 summarizes the performance of SAT relative to the sparse GNN it uses to extract the subgraph representations, across different GNNs. We observe that both variations of SAT consistently bring large performance gains to its base GNN counterpart, making it a systematic enhancer of any GNN model. Furthermore, PNA, which is the most expressive GNN we considered, has consistently the best performance when used with SAT, empirically validating our theoretical finding in Section 4.4. k-subgraph SAT also outperforms or performs equally as k-subtree SAT in almost all the cases, showing its superior expressiveness.

5.4. Hyperparameter Studies

While Table 3 showcases the added value of the SAT relative to sparse GNNs, we now dissect the components of

Table 3: Since SAT uses a GNN to extract structures, we compare the performance of the original sparse GNN to SAT which uses that GNN ("base GNN"). Across different choices of GNNs, we observe that both k-subtree and k-subgraph SATs always outperform the original sparse GNN it uses. The evaluation metrics are the same as in Table 1.

		ZINC↓		**CLUSTER↑**	**PATTERN↑**
		W/ EDGE ATTR.	W/O EDGE ATTR.	ALL	ALL
GCN	BASE GNN	0.192±0.015	0.367±0.011	68.498±0.976	71.892±0.334
	K-SUBTREE SAT	0.127±0.010	**0.174±0.009**	77.247±0.094	86.749±0.065
	K-SUBGRAPH SAT	**0.114±0.005**	0.184±0.002	**77.682±0.098**	**86.816±0.028**
GIN	BASE GNN	0.209±0.009	0.387±0.015	64.716±1.553	85.590±0.011
	K-SUBTREE SAT	0.115±0.005	0.166±0.007	77.255±0.085	**86.759±0.022**
	K-SUBGRAPH SAT	**0.095±0.002**	**0.162±0.013**	**77.502±0.282**	86.746±0.014
GRAPHSAGE	BASE GNN	–	0.398±0.002	63.844±0.110	50.516±0.001
	K-SUBTREE SAT	–	**0.164±0.004**	77.592±0.074	86.818±0.043
	K-SUBGRAPH SAT	–	0.168±0.005	**77.657±0.185**	**86.838±0.010**
PNA	BASE GNN	0.188±0.004	0.320±0.032	67.077±0.977	86.567±0.075
	K-SUBTREE SAT	0.102±0.005	0.147±0.001	77.751±0.121	**86.865±0.043**
	K-SUBGRAPH SAT	**0.094±0.008**	**0.131±0.002**	**77.856±0.104**	86.848±0.037

SAT on the ZINC dataset to identify which aspects of the architecture bring the biggest performance gains.

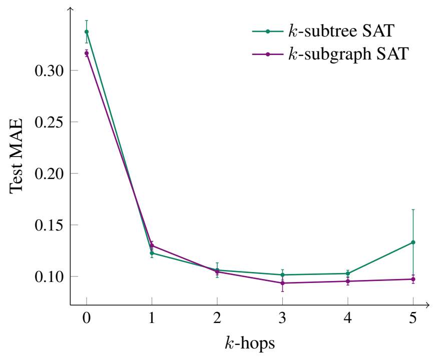
Effect of k in SAT The key contribution of SAT is its ability to explicitly incorporate structural information in the self-attention. Here, we seek to demonstrate that this information provides crucial predictive information, and study how the choice of k affects the results. Figure 3a shows how the test MAE is impacted by varying k for k-subtree and k-subgraph extractors using PNA on the ZINC dataset. All models use the RWPE. $k = 0$ corresponds to the vanilla Transformer only using absolute positional encoding, *i.e.* not using structure. We find that incorporating structural information leads to substantial improvement in performance, with optimal performance around $k = 3$ for both k-subtree and k-subgraph extractors. As k increases beyond $k = 4$, the performance in k-subtree extractors deteriorated, which is consistent with the observed phenomenon that GNNs work best in shallower networks (Kipf & Welling, 2017). We observe that k-subgraph does not suffer as much from this issue, underscoring a new aspect of its usefulness. On the other hand, k-subtree extractors are more computationally efficient and scalable to larger OGB datasets.

Effect of absolute encoding We assess here whether the absolute encoding brought complementary information to SAT. In Figure 3b, we conduct an ablation study showing the results of SAT with and without absolute positional encoding, including RWPE and Laplacian PE (Dwivedi et al., 2020). Our SAT with a positional encoding outperforms its counterpart without it, confirming the complementary nature of the two encodings. However, we also note that the performance gain brought by the absolute encoding is far less than the gain obtained by using our structure-aware attention, as shown in Figure 3a (comparing the instance of $k = 0$ to $k > 0$), emphasizing that our structure-aware attention is the more important aspect of the model.

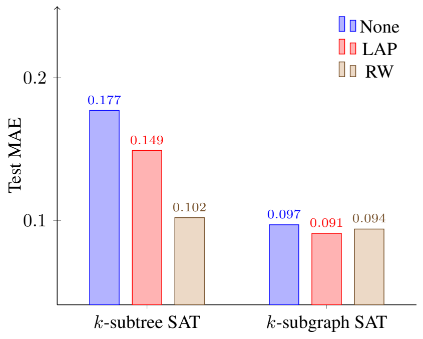
Comparison of readout methods Finally, we compare the performance of SAT models using different readout methods for aggregating node-level representations on the ZINC dataset in Figure 3c, including the CLS pooling discussed in Section 4.2. Unlike the remarkable influence of the readout method in GNNs (Xu et al., 2019), we observe very little impact in SAT models.

5.5. Model Interpretation

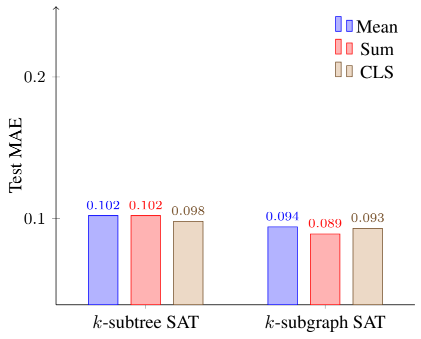
In addition to performance improvement, we show that SAT offers better model interpretability compared to the classic Transformer with only absolute positional encoding. We respectively train a SAT model and a Transformer with a CLS readout on the Mutagenicity dataset, and visualize the attention scores between the [CLS] node and other nodes learned by SAT and the Transformer in Figure 4. The salient difference between the two models is that SAT has structure-aware node embeddings, and thus we can attribute the following interpretability gains to that. While both models manage to identify some chemical motifs known for mutagenicity, such as NO_2 and NH_2, the attention scores learned by SAT are sparser and more informative, meaning that SAT puts more attention weights on these known mutagenic motifs than the Transformer with RWPE. The vanilla Transformer even fails to put attention on some important atoms such as the H atoms in the NH_2 group. The only H atoms highlighted by SAT are those in the NH_2 group, suggesting that our SAT indeed takes the structure into account. More focus on these discriminative motifs makes the SAT model less influenced by other chemical patterns that commonly exist in the dataset, such as benzene, and thus leads to overall improved performance. More results are provided in the Appendix.



(a) Effect of k



(b) Effect of absolute encoding



(c) Effect of readout method

Figure 3: We provide an analysis of the different drivers of performance in SAT on the ZINC dataset (lower is better). In Figure 3a, we show how changing the size of k affects performance (k=0 is equivalent to a vanilla Transformer that is not structure-aware). Figure 3b shows the effect of different absolute encoding methods, and Figure 3c shows the effect of different readout methods.

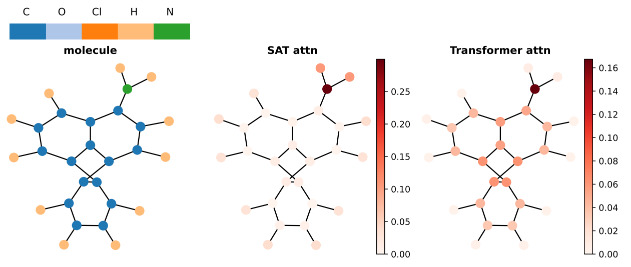


Figure 4: Attention visualization of SAT and the Transformer. The center column shows the attention weights of the [CLS] node learned by our SAT model and the right column shows the attention weights learned by the classic Transformer with the random walk positional encoding (RWPE).

6. Discussion

We introduced the SAT model, which successfully incorporates structural information into the Transformer architecture and overcomes the limitations of the absolute encoding. In addition to SOTA empirical performance with minimal hyperparameter tuning, SAT also provides better interpretability than the Transformer.

Limitations As mentioned above, k-subgraph SAT has higher memory requirements than k-subtree SAT, which can restrict its applicability if access to high memory GPUs is restricted. We see the main limitation of SAT is that it suffers from the same drawbacks as the Transformer, namely the quadratic complexity of the self-attention computation.

Future work Because SAT can be combined with any GNN, a natural extension of our work is to combine SAT with structure extractors which have shown to be strictly more expressive than the 1-WL test, such as the recent topological GNN introduced by Horn et al. (2021). Additionally, the SAT framework is flexible and can incorporate any structure extractor which produces structure-aware node representations, and could even be extended beyond using GNNs, such as differentiable graph kernels.

Another important area for future work is to focus on reducing the high memory cost and time complexity of the self-attention computation, as is being done in recent efforts for developing a so-called linear transformer, which has linear complexity in both time and space requirements (Tay et al., 2020; Wang et al., 2020; Qin et al., 2022).

Acknowledgements

This work was supported in part by the Alfried Krupp Prize for Young University Teachers of the Alfried Krupp von Bohlen und Halbach-Stiftung (K.B.). The authors would also like to thank Dr. Bastian Rieck and Dr. Carlos Oliver for their insightful feedback on the manuscript, which greatly improved it.

References

Abbe, E. Community detection and stochastic block models: Recent developments. *Journal of Machine Learning Research (JMLR)*, 18(177):1–86, 2018. (Cited on 17)

Alon, U. and Yahav, E. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations (ICLR)*, 2021. (Cited on 1)

Alsentzer, E., Finlayson, S. G., Li, M. M., and Zitnik, M. Subgraph neural networks. In *Proceedings of Neural*

Information Processing Systems, NeurIPS, 2020. (Cited on 21)

Bodnar, C., Frasca, F., Otter, N., Wang, Y. G., Liò, P., Montufar, G. F., and Bronstein, M. Weisfeiler and lehman go cellular: Cw networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. (Cited on 5)

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. (Cited on 1)

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. Principal neighbourhood aggregation for graph nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (Cited on 2, 7)

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2020. (Cited on 1)

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. In *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021. (Cited on 2, 3, 7)

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020. (Cited on 7, 8, 16, 17)

Dwivedi, V. P., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2022. (Cited on 2, 3, 5, 7, 17)

Fan, W., Ma, Y., Li, Q., He, Y., Zhao, E., Tang, J., and Yin, D. Graph neural networks for social recommendation. In *The World Wide Web Conference*, 2019. (Cited on 1)

Gao, B. and Pavel, L. On the properties of the softmax function with application in game theory and reinforcement learning. *arXiv preprint arXiv:1704.00805*, 2017. (Cited on 14)

Gao, H. and Ji, S. Graph u-nets. In *International Conference on Machine Learning*, pp. 2083–2092, 2019. (Cited on 21)

Gaudelet, T., Day, B., Jamasb, A. R., Soman, J., Regep, C., Liu, G., Hayter, J. B., Vickers, R., Roberts, C., Tang, J., et al. Utilizing graph machine learning within drug discovery and development. *Briefings in Bioinformatics*, 22(6):bbab159, 2021. (Cited on 1)

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017. (Cited on 1, 2)

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. (Cited on 2, 7)

Horn, M., De Brouwer, E., Moor, M., Moreau, Y., Rieck, B., and Borgwardt, K. Topological graph neural networks. 2021. (Cited on 9)

Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991. (Cited on 6, 15)

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020a. (Cited on 7, 16, 17)

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., and Leskovec, J. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations (ICLR)*, 2020b. (Cited on 2)

Ingraham, J., Garg, V., Barzilay, R., and Jaakkola, T. Generative models for graph-based protein design. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. (Cited on 1)

Irwin, J. J., Sterling, T., Mysinger, M. M., Bolstad, E. S., and Coleman, R. G. Zinc: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52(7):1757–1768, 2012. (Cited on 16)

Jain, P., Wu, Z., Wright, M., Mirhoseini, A., Gonzalez, J. E., and Stoica, I. Representing long-range context for graph neural networks with global attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. (Cited on 3, 5, 7)

Kersting, K., Kriege, N. M., Morris, C., Mutzel, P., and Neumann, M. Benchmark data sets for graph kernels, 2016. `http://graphkernels.cs.tu-dortmund.de`. (Cited on 20)

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. (Cited on 2, 7, 8)

Kreuzer, D., Beaini, D., Hamilton, W. L., Létourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. (Cited on 2, 7)

Li, G., Müller, M., Thabet, A., and Ghanem, B. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2019. (Cited on 1)

Li, G., Xiong, C., Thabet, A., and Ghanem, B. Deepergcn: All you need to train deeper gcns, 2020a. (Cited on 2, 7)

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding: Design provably more powerful neural networks for graph representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020b. (Cited on 2)

Li, Q., Han, Z., and Wu, X. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. (Cited on 1)

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2016. (Cited on 17)

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2018. (Cited on 17)

Mesquita, D., Souza, A. H., and Kaski, S. Rethinking pooling in graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (Cited on 21)

Mialon, G., Chen, D., Selosse, M., and Mairal, J. Graphit: Encoding graph structure in transformers, 2021. (Cited on 3, 5)

Micchelli, C. A., Xu, Y., and Zhang, H. Universal kernels. *Journal of Machine Learning Research (JMLR)*, 7(12), 2006. (Cited on 16)

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019. (Cited on 1)

Nikolentzos, G. and Vazirgiannis, M. Random walk graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (Cited on 5)

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations (ICLR)*, 2020. (Cited on 1)

Qin, Z., Sun, W., Deng, H., Li, D., Wei, Y., Lv, B., Yan, J., Kong, L., and Zhong, Y. cosformer: Rethinking softmax in attention. In *International Conference on Learning Representations*, 2022. (Cited on 9)

Rives, A., Meier, J., Sercu, T., Goyal, S., Lin, Z., Liu, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 118(15), 2021. (Cited on 1)

Rong, Y., Bian, Y., Xu, T., Xie, W., Wei, Y., Huang, W., and Huang, J. Self-supervised graph transformer on large-scale molecular data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. (Cited on 3)

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018. (Cited on 2)

Shi, Y., Huang, Z., Feng, S., Zhong, H., Wang, W., and Sun, Y. Masked label prediction: Unified message passing model for semi-supervised classification. In Zhou, Z.-H. (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)*, pp. 1548–1554. International Joint Conferences on Artificial Intelligence Organization, 8 2021. (Cited on 3)

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020. (Cited on 9)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. (Cited on 1, 2, 3, 17)

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*, 2018. (Cited on 2, 7)

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity, 2020. (Cited on 9)

Wijesinghe, A. and Wang, Q. A new perspective on "how graph neural networks go beyond weisfeiler-lehman?". In *International Conference on Learning Representations*, 2022. (Cited on 4)

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations (ICLR)*, 2019. (Cited on 1, 2, 4, 6, 7, 8, 13, 15)

Yang, M., Wang, R., Shen, Y., Qi, H., and Yin, B. Breaking the expression bottleneck of graph neural networks. *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2022. doi: 10.1109/TKDE.2022.3168070. (Cited on 2, 7)

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. (Cited on 3, 7)

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018. (Cited on 21)

You, J., Ying, R., and Leskovec, J. Position-aware graph neural networks. In *International Conference on Machine Learning (ICML)*, 2019. (Cited on 2)

Zhang, J., Zhang, H., Xia, C., and Sun, L. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020. (Cited on 2)

Zhang, M. and Li, P. Nested graph neural networks. In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS)*, 2021. (Cited on 4, 5)

Appendix

This appendix provides both theoretical and experimental materials and is organized as follows: Section A provides a more detailed background on graph neural networks. Section B presents proofs of Theorem 1 and 2. Section C provides experimental details and additional results. Section D provides details on the model interpretation and additional visualization results.

A. Background on Graph Neural Networks

The overarching idea of a graph neural network is to iteratively update a node's embedding by incorporating information sent from its neighbors. Xu et al. (2019) provide a general framework of the steps incorporated in this process by generalizing the different frameworks into `AGGREGATE`, `COMBINE` and `READOUT` steps. The various flavors of GNNs can be typically understood as variations within these three functions. For a given layer l, the `AGGREGATE` step aggregates (e.g. using the sum or mean) the representations of the neighbors of a given node, which is then combined with the given node's representation from the previous layer in the `COMBINE` step. This is followed by a non-linear function, such as ReLU, and the updated node representations are then passed to the next layer. These two steps are repeated for as many layers as there are in the network. It is worth noting that the output of these two steps provides representations of nodes which accounts for local sub-structures of size only increased by one, which would thus require a very deep network to capture interactions between the given node and all other nodes (the depth should not be smaller than the diameter of the graph). At the end of the network, the `READOUT` function provides a pooling function to convert the representations to the appropriate output-level granularity (e.g. node-level or graph-level). Both the `AGGREGATE` and `READOUT` steps must be invariant to node permutations.

B. Theoretical Analysis

B.1. Controllability of the Representations from the Structure-Aware Attention

Theorem 1. *Assume that f is a Lipschitz mapping with the Lipschitz constant denoted by $Lip(f)$ and the structure extractor φ is bounded by a constant C_φ on the space of subgraphs. For any pair of nodes v and v' in two graphs $G = (V, E, \mathbf{X})$ and $G' = (V', E', \mathbf{X}')$ with the same number of nodes $|V| = |V'| = n$, the distance between their representations after the structure-aware attention is bounded by:*

$$\|SA\text{-}attn(v) - SA\text{-}attn(v')\| \leq C_1[\|h_v - h'_{v'}\| + D(\mathbf{H}, \mathbf{H}')] + C_2 D(\mathbf{X}, \mathbf{X}'), \tag{11}$$

where $h_w := \varphi(w, G)$ denotes the subgraph representation at node w for any $w \in V$ and $h'_{w'} := \varphi(w', G')$ similarly, and $\mathbf{H} = (h_w)_{w\in V}$ and $\mathbf{H}' = (h'_{w'})_{w'\in V'}$ denote the multiset of subgraph representations in G and G' respectively. Denoting by $\Pi(V, V')$ the set of permutations between V and V', D is a matching metric between two multisets of representations with the same cardinality, defined as

$$D(\mathbf{X}, \mathbf{X}') := \inf_{\pi\in\Pi(V,V')} \sup_{w\in V} \|x_w - x'_{\pi(w)}\|.$$

C_1 and C_2 are constants given by:

$$C_1 = \sqrt{\frac{2}{d_{out}}} n Lip(f) C_\varphi \|\mathbf{W_Q}\|_\infty \|\mathbf{W_K}\|_\infty, \quad C_2 = Lip(f).$$

Proof. Let us denote by

$$z_v = (\langle \mathbf{W_Q} h_v, \mathbf{W_K} h_w\rangle)_{w\in V} \in \mathbb{R}^n,$$
$$z'_{v'} = (\langle \mathbf{W_Q} h'_{v'}, \mathbf{W_K} h'_{w'}\rangle)_{w'\in V'} \in \mathbb{R}^n,$$

and by $\mathrm{softmax}(z) \in \mathbb{R}^n$ for any $z \in \mathbb{R}^n$ with its i-th coefficient

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i/\sqrt{d_{out}})}{\sum_{j=1}^n \exp(z_j/\sqrt{d_{out}})}.$$

Then, we have

$$
\begin{aligned}
&\|\text{SA-Attn}(v) - \text{SA-Attn}(v')\| \\
=&\left\| \sum_{w\in V} \operatorname{softmax}(z_v)_w f(x_w) - \sum_{w'\in V'} \operatorname{softmax}(z'_{v'})_{w'} f(x'_{w'}) \right\| \\
=&\left\| \sum_{w\in V} (\operatorname{softmax}(z_v)_w - \operatorname{softmax}(z'_{v'})_{\pi(w)}) f(x_w) + \sum_{w\in V} \operatorname{softmax}(z'_{v'})_{\pi(w)} f(x_w) - \sum_{w'\in V'} \operatorname{softmax}(z'_{v'})_{w'} (f(x'_{w'})) \right\| \\
\leq&\left\| \sum_{w\in V} (\operatorname{softmax}(z_v)_w - \operatorname{softmax}(z'_{v'})_{\pi(w)}) f(x_w) \right\| + \left\| \sum_{w'\in V'} \operatorname{softmax}(z'_{v'})_{w'} (f(x_{\pi^{-1}(w')}) - f(x'_{w'})) \right\|
\end{aligned}
$$

where $\pi : V \to V'$ is an arbitrary permutation and we used the triangle inequality. Now we need to bound the two terms respectively. We first bound the second term:

$$
\begin{aligned}
\left\| \sum_{w'\in V'} \operatorname{softmax}(z'_{v'})_{w'} (f(x_{\pi^{-1}(w')}) - f(x'_{w'})) \right\| &\leq \sum_{w'\in V'} \operatorname{softmax}(z'_{v'})_{w'} \left\| f(x_{\pi^{-1}(w')}) - f(x'_{w'}) \right\| \\
&\leq \sum_{w'\in V'} \operatorname{softmax}(z'_{v'})_{w'} \text{Lip}(f) \|x_{\pi^{-1}(w')} - x'_{w'}\| \\
&= \text{Lip}(f) \sum_{w'\in V'} \operatorname{softmax}(z'_{v'})_{w'} \|x_{\pi^{-1}(w')} - x'_{w'}\| \\
&\leq \text{Lip}(f) \sup_{w'\in V'} \|x_{\pi^{-1}(w')} - x'_{w'}\| \\
&= \text{Lip}(f) \sup_{w\in V} \|x_w - x'_{\pi(w)}\|
\end{aligned}
$$

where the first inequality is a triangle inequality, the second inequality uses the Lipschitzness of f. And for the first term, we can upper-bound it by

$$
\begin{aligned}
&\left\| \sum_{w\in V} (\operatorname{softmax}(z_v)_w - \operatorname{softmax}(z'_{v'})_{\pi(w)}) f(x_w) \right\| \\
\leq&\|\operatorname{softmax}(z_v) - \operatorname{softmax}((z'_{v'})_{\pi})\| \sqrt{\sum_{w\in V} \|f(x_w)\|^2} \\
\leq&\frac{1}{\sqrt{d_{out}}} \|z_v - (z'_{v'})_{\pi}\| \sqrt{n} \text{Lip}(f),
\end{aligned}
$$

where by abuse of notation, $(z)_\pi \in \mathbb{R}^n$ denotes the vector whose w-th entry is $z_{\pi(w)}$ for any $z \in \mathbb{R}^n$. The first inequality comes from a simple matrix norm inequality, and the second inequality uses the fact that $\operatorname{softmax}$ function is $1/\sqrt{d_{out}}$-Lipschitz (see *e.g.* Gao & Pavel (2017)). Then, we have

$$
\begin{aligned}
\|z_v - (z'_{v'})_\pi\|^2 &= \sum_{w\in V} \left(\langle \mathbf{W_Q} h_v, \mathbf{W_K} h_w \rangle - \langle \mathbf{W_Q} h'_{v'}, \mathbf{W_K} h'_{\pi(w)} \rangle \right)^2 \\
&= \sum_{w\in V} \left(\langle \mathbf{W_Q} h_v, \mathbf{W_K} (h_w - h'_{\pi(w)}) \rangle + \langle \mathbf{W_Q} (h_v - h'_{v'}), \mathbf{W_K} h'_{\pi(w)} \rangle \right)^2 \\
&\leq 2 \sum_{w\in V} \left(\langle \mathbf{W_Q} h_v, \mathbf{W_K} (h_w - h'_{\pi(w)}) \rangle^2 + \langle \mathbf{W_Q} (h_v - h'_{v'}), \mathbf{W_K} h'_{\pi(w)} \rangle^2 \right) \\
&\leq 2 \sum_{w\in V} \left(\|\mathbf{W_Q} h_v\|^2 \|\mathbf{W_K} (h_w - h'_{\pi(w)})\|^2 + \|\mathbf{W_Q} (h_v - h'_{v'})\|^2 \|\mathbf{W_K} h'_{\pi(w)}\|^2 \right) \\
&\leq 2 \sum_{w\in V} \left(C_\varphi^2 \|\mathbf{W_Q}\|_\infty^2 \|\mathbf{W_K}\|_\infty^2 \|h_w - h'_{\pi(w)}\|^2 + \|\mathbf{W_Q}\|_\infty^2 \|h_v - h'_{v'}\|^2 C_\varphi^2 \|\mathbf{W_K}\|_\infty^2 \right) \\
&\leq 2n C_\varphi^2 \|\mathbf{W_Q}\|_\infty^2 \|\mathbf{W_K}\|_\infty^2 \left(\|h_v - h'_{v'}\|^2 + \sup_{w\in V} \|h_w - h'_{\pi(w)}\|^2 \right),
\end{aligned}
$$

where the first inequality comes from $(a+b)^2 \leq 2(a^2+b^2)$, the second one uses the Cauchy-Schwarz inequality and the third one uses the definition of spectral norm and the bound of the structure extractor function. Then, we obtain the following inequality

$$\begin{aligned}
&\left\| \sum_{w \in V} (\text{softmax}(z_v)_w - \text{softmax}(z'_{v'})_{\pi(w)}) f(x_w) \right\| \\
\leq & \sqrt{\frac{2}{d_{out}}} n \text{Lip}(f) C_\varphi \|\mathbf{W_Q}\|_\infty \|\mathbf{W_K}\|_\infty \left(\|h_v - h'_{v'}\| + \sup_{w \in V} \|h_w - h'_{\pi(w)}\| \right)
\end{aligned}$$

By combining the upper bounds of the first and the second term, we obtain an upper bound for the distance between the structure-aware attention representations:

$$\|\text{SA-attn}(v) - \text{SA-attn}(v')\| \leq C_1 \left(\|h_v - h'_{v'}\| + \sup_{w \in V} \|h_w - h'_{\pi(w)}\| \right) + C_2 \sup_{w \in V} \|x_w - x'_{\pi(w)}\|,$$

for any permutation $\pi \in \Pi(V, V')$, where

$$\begin{aligned}
C_1 &= \sqrt{\frac{2}{d_{out}}} n \text{Lip}(f) C_\varphi \|\mathbf{W_Q}\|_\infty \|\mathbf{W_K}\|_\infty \\
C_2 &= \text{Lip}(f).
\end{aligned}$$

Finally, by taking the infimum over the set of permutations, we obtain the inequality in the theorem. □

B.2. Expressivity Analysis

Here, we assume that f can be any continuous mapping and it is approximated by an MLP network through the universal approximation theorem (Hornik, 1991) in practice.

Theorem 2. *Assume that the space of node attributes $\mathcal{X}$ is countable. For any pair of nodes v and v' in two graphs $G = (V, E, \mathbf{X})$ and $G' = (V', E', \mathbf{X}')$, assume that there exists a node u_1 in V such that $x_{u_1} \neq x_w$ for any $w \in V$ and a node u_2 in V such that its subgraph representation $\varphi(u_2, G) \neq \varphi(w, G)$ for any $w \in V$. Then, there exists a set of parameters and a mapping $f : \mathcal{X} \to \mathbb{R}^{d_{out}}$ such that their representations after the structure-aware attention are different, i.e. SA-attn$(v) \neq$ SA-attn(v'), if their subgraph representations are different, i.e. $\varphi(v, G) \neq \varphi(v', G')$.*

Proof. This theorem amounts to showing the injectivity of the original dot-product attention with respect to the query, that is to show

$$\text{Attn}(h_v, x_v, G) = \sum_{u \in V} \frac{\kappa_{\exp}(h_v, h_u)}{\sum_{w \in V} \kappa_{\exp}(h_v, h_w)} f(x_u)$$

is injective in h_v, where

$$\kappa_{\exp}(h, h') := \exp\left(\langle \mathbf{W_Q} h + b_Q, \mathbf{W_K} h' + b_K \rangle / \sqrt{d_{out}} \right). \tag{12}$$

Here we consider the offset terms that were omitted in Eq. (1). Let us prove the contrapositive of the theorem. We assume that $\text{Attn}(h_v, x_v, G) = \text{Attn}(h'_{v'}, x'_{v'}, G')$ for any set of parameters and any mapping f and want to show that $h_v = h'_{v'}$.

Without loss of generality, we assume that G and G' have the same number of nodes, that is $|V| = |V'| = n$. Otherwise, one can easily add some virtual isolated nodes to the smaller graph. Now if we take $\mathbf{W_Q} = \mathbf{W_K} = 0$, all the softmax coefficients will be identical and we have

$$\sum_{w \in V} f(x_w) = \sum_{w' \in V'} f(x'_{w'}).$$

Thus, by Lemma 5 of Xu et al. (2019), there exists a mapping f such that the multisets X and X' are identical.

As a consequence, we can re-enumerate the nodes in two graphs by a sequence V (by abuse of notation, we keep using V here) such that $x_u = x'_u$ for any $u \in V$. Then, we can rewrite the equality $\text{Attn}(h_v, x_v, G) = \text{Attn}(h'_{v'}, x'_{v'}, G')$ as

$$\sum_{u \in V} \left(\frac{\kappa_{\exp}(h_v, h_u)}{\sum_{w \in V} \kappa_{\exp}(h_v, h_w)} - \frac{\kappa_{\exp}(h'_{v'}, h'_u)}{\sum_{w \in V} \kappa_{\exp}(h'_{v'}, h'_w)} \right) f(x_u) = 0.$$

Now since there exists a node u_1 in V such that its attributes are different from all other nodes, *i.e.* $x_{u_1} \neq x_w$ for any $w \in V$, we can find a mapping f such that $f(x_{u_1})$ is not in the span of $(f(x_w))_{w\in V, w\neq u_1}$. Then, by their independence we have

$$\frac{\kappa_{\exp}(h_v, h_{u_1})}{\sum_{w\in V}\kappa_{\exp}(h_v, h_w)} = \frac{\kappa_{\exp}(h'_{v'}, h'_{u_1})}{\sum_{w\in V}\kappa_{\exp}(h'_{v'}, h'_w)},$$

for any $\mathbf{W_Q}$, $\mathbf{W_K}$, b_Q and b_K.

On the one hand, if we take $\mathbf{W_Q} = 0$, we have for any $\mathbf{W_K}$, b_Q and b_K that

$$\frac{\exp\left(\langle b_Q, \mathbf{W_K} h_{u_1} + b_K\rangle/\sqrt{d_{out}}\right)}{\sum_{w\in V}\exp\left(\langle b_Q, \mathbf{W_K} h_w + b_K\rangle/\sqrt{d_{out}}\right)} = \frac{\exp\left(\langle b_Q, \mathbf{W_K} h'_{u_1} + b_K\rangle/\sqrt{d_{out}}\right)}{\sum_{w\in V}\exp\left(\langle b_Q, \mathbf{W_K} h'_w + b_K\rangle/\sqrt{d_{out}}\right)}.$$

On the other hand if we take $b_Q = 0$ we have for any $\mathbf{W_Q}$, $\mathbf{W_K}$ and b_K that

$$\begin{aligned}\frac{\exp\left(\langle \mathbf{W_Q} h_v, \mathbf{W_K} h_{u_1} + b_K\rangle/\sqrt{d_{out}}\right)}{\sum_{w\in V}\exp\left(\langle \mathbf{W_Q} h_v, \mathbf{W_K} h_w + b_K\rangle/\sqrt{d_{out}}\right)} &= \frac{\exp\left(\langle \mathbf{W_Q} h'_{v'}, \mathbf{W_K} h'_{u_1} + b_K\rangle/\sqrt{d_{out}}\right)}{\sum_{w\in V}\exp\left(\langle \mathbf{W_Q} h'_{v'}, \mathbf{W_K} h'_w + b_K\rangle/\sqrt{d_{out}}\right)} \\ &= \frac{\exp\left(\langle \mathbf{W_Q} h'_{v'}, \mathbf{W_K} h_{u_1} + b_K\rangle/\sqrt{d_{out}}\right)}{\sum_{w\in V}\exp\left(\langle \mathbf{W_Q} h'_{v'}, \mathbf{W_K} h_w + b_K\rangle/\sqrt{d_{out}}\right)},\end{aligned}$$

where the second equality is obtained by replacing b_Q with $\mathbf{W_Q} h'_{v'}$ in the above equality. Then, we can rewrite the above equality as below:

$$\sum_{w\in V}\exp\left(\frac{\langle \mathbf{W_Q} h_v, \mathbf{W_K}(h_w - h_{u_1})\rangle}{\sqrt{d_{out}}}\right) = \sum_{w\in V}\exp\left(\frac{\langle \mathbf{W_Q} h'_{v'}, \mathbf{W_K}(h_w - h_{u_1})\rangle}{\sqrt{d_{out}}}\right).$$

If we denote by $\phi : \mathbb{R}^{d_{out}} \to \mathcal{H}$ the feature mapping associated with the dot product kernel $\kappa_{\exp}(t, t') = \exp(\langle t, t'\rangle/\sqrt{d_{out}})$ and $\mathcal{H}$ the correspond reproducing kernel Hilbert space, we then have for any $\mathbf{W_Q}$ and $\mathbf{W_K}$ that

$$\left\langle \phi(\mathbf{W_Q} h_v) - \phi(\mathbf{W_Q} h'_{v'}), \sum_{w\in V}\phi(\mathbf{W_K}(h_w - h_{u_1}))\right\rangle_{\mathcal{H}} = 0.$$

Since by assumption there exists a $u_2 \in V$ such that $h_{u_2} - h_{u_1} \neq 0$ and $\kappa_{\exp}$ is a universal kernel (Micchelli et al., 2006), $\mathbf{W_K} \mapsto \phi(\mathbf{W_K}(h_{u_2} - h_{u_1}))$ is dense in $\mathcal{H}$ and we have $\phi(\mathbf{W_Q} h_v) = \phi(\mathbf{W_Q} h'_{v'})$. We can then conclude, by the injectivity of ϕ, that

$$\mathbf{W_Q} h_v = \mathbf{W_Q} h'_{v'},$$

for any $\mathbf{W_Q}$, and thus $h_v = h'_{v'}$. Now by taking $h_v = \varphi(v, G)$ and $h'_{v'} = \varphi(v', G')$, we obtain the theorem. □

C. Experimental Details and Additional Results

In this section, we provide implementation details and additional experimental results.

C.1. Computation Details

All experiments were performed on a shared GPU cluster equipped with GTX1080, GTX1080TI, GTX2080TI and TITAN RTX. About 20 of these GPUs were used simultaneously, and the total computational cost of this research project was about 1k GPU hours.

C.2. Datasets Description

We provide details of the datasets used in our experiments, including ZINC (Irwin et al., 2012), CLUSTER (Dwivedi et al., 2020), PATTERN (Dwivedi et al., 2020), OGBG-PPA (Hu et al., 2020a) and OGBG-CODE2 (Hu et al., 2020a). For each dataset, we follow their respective training protocols and use the standard train/validation/test splits and evaluation metrics.

ZINC. The ZINC dataset is a graph regression dataset comprised of molecules, where the task is to predict constrained solubility. Like Dwivedi et al. (2020), we use the subset of 12K molecules and follow their same splits.

Table 4: Hyperparameters for SAT models trained on different datasets. RWPE-p indicates using p steps in the random walk positional encoding, which results in a p-dimensional vector as the positional representation for each node.

Hyperparameter	**ZINC**	**CLUSTER**	**PATTERN**	**OGBG-PPA**	**OGBG-CODE2**
#Layers	6	16	6	3	4
Hidden dimensions	64	48	64	128	256
FFN hidden dimensions			$2\times$Hidden dimensions		
#Attention heads	8	8	8	8	$\{\mathbf{4}, 8\}$
Dropout			$\{0.0, 0.1, 0.2, 0.3, 0.4\}$		
Size of subgraphs k			$\{1, 2, 3, 4\}$		
Readout method	mean	None	None	mean	mean
Absolute PE	RWPE-20	RWPE-3	RWPE-7	None	None
Learning rate	0.001	0.0005	0.0003	0.0003	0.0001
Batch size	128	32	32	32	32
#Epochs	2000	200	200	200	30
Warm-up steps	5000	5000	5000	10 epochs	2 epochs
Weight decay	1e-5	1e-4	1e-4	1e-4	1e-6

PATTERN and CLUSTER. PATTERN and CLUSTER Dwivedi et al. (2020) are synthetic datasets that were created using Stochastic Block Models (Abbe, 2018). The goal for both datasets is node classification, with PATTERN focused on detecting a given pattern in the dataset, and with CLUSTER focused on identifying communities within the graphs. For PATTERN, the binary class label corresponds to whether a node is part of the predefined pattern or not; for CLUSTER, the multi-class label indicates membership in a community. We use the splits as is used in Dwivedi et al. (2020).

OGBG-PPA. PPA (Hu et al., 2020a) is comprised of protein-protein association networks where the goal is to correctly classify the network into one of 37 classes representing the category of species the network is from. Nodes represent proteins and edges represent associations between proteins. Edge attributes represent information relative to the association, such as co-expression. We use the standard splits provided by Hu et al. (2020a).

OGBG-CODE2. CODE2 (Hu et al., 2020a) is a dataset containing source code from the Python programming language. It is made up of Abstract Syntax Trees where the task is to correctly classify the sub-tokens that comprise the method name. We use the standard splits provided by Hu et al. (2020a).

C.3. Hyperparameter Choices and Reproducibility

Hyperparameter choice. In general, we perform a very limited hyperparameter search to produce the results in Table 1 and Table 2. The hyperparameters for training SAT models on different datasets are summarized in Table 4, where only the dropout rate and the size of the subgraph k are tuned ($k \in \{1, 2, 3, 4\}$). We use fixed RWPE (Dwivedi et al., 2022) with SAT on ZINC, PATTERN and CLUSTER. In all experiments, we use the validation set to select the dropout rate and the size of the subtree or subgraph $k \in \{1, 2, 3, 4\}$. All other hyperparameters are fixed for simplicity, including setting the readout method to mean pooling. We did not use RWPE on OGBG-PPA and OGBG-CODE2 as we observed very little performance improvement. Note that we only use $k = 1$ for the k-subgraph SAT models on CLUSTER and PATTERN due to its large memory requirement, which already leads to performance boost compared to the k-subtree SAT using a larger k. Reported results are the average over 4 seeds on ZINC, PATTERN and CLUSTER, as is done in Dwivedi et al. (2020), and averaged over 10 seeds on OGBG-PPA and OGBG-CODE2.

Optimization. All our models are trained with the AdamW optimizer (Loshchilov & Hutter, 2018) with a standard warm-up strategy suggested for Transformers in Vaswani et al. (2017). We use either the L1 loss or the cross-entropy loss depending on whether the task is regression or classification. The learning rate scheduler proposed in the Transformer is used on the ZINC, PATTERN and CLUSTER datasets and a cosine scheduler (Loshchilov & Hutter, 2016) is used on the larger OGBG-PPA and OGBG-CODE2 datasets.

Table 5: Number of parameters and training time per epoch for k-subtree SAT models using the hyperparameters in Table 4. Various GNNs are used as the base GNN in SAT.

	ZINC	**CLUSTER**	**PATTERN**	**OGBG-PPA**	**OGBG-CODE2**
Base GNN			#Parameters		
GCN	421k	571k	380k	766k	14,030k
GIN	495k	684k	455k	866k	14,554k
PNA	523k	741k	493k	1,088k	15,734k
Base GNN			GPU time on a single TITAN RTX/epoch		
GCN	6s	142s	40s	308s	40min
GIN	6s	144s	62s	310s	40min
PNA	9s	178s	90s	660s	55min

Table 6: Test MAE for SAT models using different structure extractors and readout methods on the ZINC dataset.

		W/O EDGE ATTRIBUTES			W/ EDGE ATTRIBUTES		
	BASE GNN	MEAN	SUM	CLS	MEAN	SUM	CLS
K-SUBTREE SAT	GCN	0.174±0.009	0.170±0.010	0.167±9.005	0.127±0.010	0.117±0.008	0.115±0.007
	GIN	0.166±0.007	0.162±0.010	0.157±0.002	0.115±0.005	0.112±0.008	0.104±0.003
	GRAPHSAGE	0.164±0.004	0.165±0.008	0.156±0.005	-	-	-
	PNA	0.147±0.001	0.142±0.008	**0.135±0.004**	0.102±0.005	0.102±0.003	**0.098±0.008**
K-SUBGRAPH SAT	GCN	0.184±0.002	0.186±0.007	0.184±0.007	0.114±0.005	0.103±0.002	0.103±0.008
	GIN	0.162±0.013	0.158±0.007	0.162±0.005	0.095±0.002	0.097±0.002	0.098±0.010
	GRAPHSAGE	0.168±0.005	0.165±0.005	0.169±0.005	-	-	-
	PNA	0.131±0.002	0.129±0.003	**0.128±0.004**	0.094±0.008	**0.089±0.002**	0.093±0.009

Number of parameters and computation time. In Table 5, we report the number of parameters and the training time per epoch for SAT with k-subtree GNN extractors using the hyperparameters selected from Table 4. Note that the number of parameters used in our SAT on OGB datasets is smaller than most of the state-of-art methods.

C.4. Additional Results

We provide additional experimental results on ZINC, OGBG-PPA and OGBG-CODE2.

C.4.1. ADDITIONAL RESULTS ON ZINC

We report a more thorough comparison of SAT instances using different structure extractors and different readout methods in Table 6. We find that SAT models with PNA consistently outperform other GNNs. Additionally, the readout methods have very little impact on the prediction performance.

C.4.2. ADDITIONAL RESULTS ON OGBG-PPA

Table 7 summarizes the results for k-subtree SAT with different GNNs compared to state-of-the-art methods on OGBG-PPA. All the results are computed from 10 runs using different random seeds.

C.4.3. ADDITIONAL RESULTS ON OGBG-CODE2

Table 8 summarizes the results for k-subtree SAT with different GNNs compared to state-of-the-art methods on OGBG-CODE2. All the results are computed from 10 runs using different random seeds.

Table 7: Comparison of SAT and SOTA methods on the OGBG-PPA dataset. All results are computed from 10 different runs.

	OGBG-PPA	
METHOD	TEST ACCURACY	VALIDATION ACCURACY
GCN	0.6839±0.0084	0.6497±0.0034
GCN-VIRTUAL NODE	0.6857±0.0061	0.6511±0.0048
GIN	0.6892±0.0100	0.6562±0.0107
GIN-VIRTUAL NODE	0.7037±0.0107	0.6678±0.0105
TRANSFORMER	0.6454±0.0033	0.6221±0.0039
k-SUBTREE SAT-GCN	0.7483±0.0048	**0.7072±0.0030**
k-SUBTREE SAT-GIN	0.7306±0.0076	0.6928±0.0058
k-SUBTREE SAT-PNA	**0.7522±0.0056**	0.7025±0.0064

Table 8: Comparison of SAT and SOTA methods on the OGBG-CODE2 dataset. All results are computed from 10 different runs.

	OGBG-CODE2	
METHOD	TEST F1 SCORE	VALIDATION F1 SCORE
GCN	0.1507±0.0018	0.1399±0.0017
GCN-VIRTUAL NODE	0.1581±0.0026	0.1461±0.0013
GIN	0.1495±0.0023	0.1376±0.0016
GIN-VIRTUAL NODE	0.1581±0.0026	0.1439±0.0020
TRANSFORMER	0.1670±0.0015	0.1546±0.0018
GRAPHTRANS	0.1830±0.0024	0.1661±0.0012
k-SUBTREE SAT-GCN	0.1934±0.0020	**0.1777±0.0011**
k-SUBTREE SAT-GIN	0.1910±0.0023	0.1748±0.0016
k-SUBTREE SAT-PNA	**0.1937±0.0028**	0.1773±0.0023

D. Model Interpretation

In this section, we provide implementation details about the model visualization.

D.1. Dataset and Training Details

We use the Mutagenicity dataset (Kersting et al., 2016), consisting of 4337 molecular graphs labeled based on their mutagenic effect. We randomly split the dataset into train/val/test sets in a stratified way with a proportion of 80/10/10. We first train a two-layer vanilla Transformer model using RWPE. The hidden dimension and the number of heads are fixed to 64 and 8 respectively. The CLS pooling as described in Section 4.2 is chosen as the readout method for visualization purpose. We also train a k-subtree SAT using exactly the same hyperparameter setting except that *it does not use any absolute positional encoding*. k is fixed to 2. For both models, we use the AdamW optimizer and the optimization strategy described in Section C.3. We train enough epochs until both models converge. While the classic Transformer with RWPE achieves a test accuracy of 78%, the k-subtree SAT achieves a 82% test accuracy.

D.2. Additional Results

Visualization of attention scores. Here, we provide additional visualization examples of attention scores of the [CLS] node from the Mutagenicity dataset, learned by SAT and a vanilla Transformer. Figure 5 provides several examples of attention learned weights. SAT generally learns sparser and more informative weights even for very large graph as shown in the left panel of the middle row.

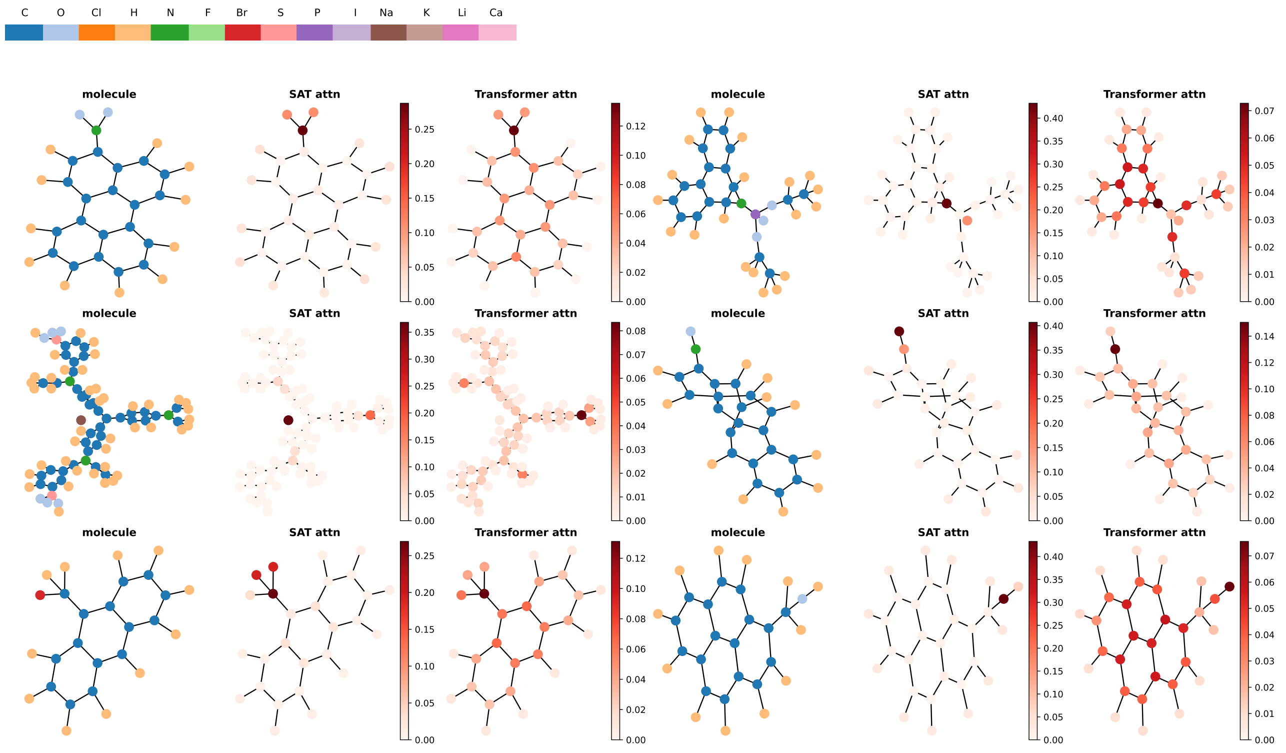


Figure 5: Attention visualization of SAT and the Transformer. The middle column shows the attention weights of the [CLS] node learned by our SAT model and the right column shows the attention weights learned by the classic Transformer with RWPE.

E. Relationship to Subgraph Neural Networks and Graph Pooling

In the following we clarify the relationship (and differences) of SAT to Subgraph Neural Networks (Alsentzer et al., 2020) as well as to the general topic of graph pooling.

E.1. Differences to Subgraph Neural Networks

Subgraph Neural Networks (SNN) (Alsentzer et al., 2020) explores explicitly incorporating position, neighborhood and structural information, for the purpose of solving the problem of subgraph prediction. SNN generates representations at the level of subgraph (rather than node). SAT, on the other hand, is instead motivated by modeling the *structural interaction* (through the dot-product attention) between nodes in the Transformer architecture by generating node representations that are structure-aware. This structure-aware aspect is achieved via a structure extractor, which can be any function that extracts local structural information for a given node and does not necessarily need to explicitly extract subgraphs. For example, the k-subtree GNN extractor does not explicitly extract the subgraph, but rather only uses the node representation generated from a GNN. This aspect also makes the k-subtree SAT very scalable. In contrast to SNN, the input to the resulting SAT is not subgraphs but rather the original graph, and the structure-aware node representations are computed as the query and key for the dot-product attention at each layer.

E.2. Relationship to Graph Pooling

In GNNs, structural information is traditionally incorporated into node embeddings via the neighborhood aggregation process. A supplemental way to incorporate structural information is through a process called *local pooling*, which is typically based on graph clustering (Ying et al., 2018). Local pooling coarsens the adjacency matrix at layer l in the network by, for example, applying a clustering algorithm to cluster nodes, and then replacing the adjacency matrix with the cluster assignment matrix, where all nodes within a cluster are connected by an edge. An alternative approach to the pooling layer is based on sampling nodes (Gao & Ji, 2019). While Mesquita et al. (2020) found that these local pooling operations currently do not improve performance relative to more simple operations, local pooling could in theory be incorporated into the existing SAT's k-subtree and k-subgraph GNN extractors, as it is another layer in a GNN.