SEA: Graph Shell Attention in Graph Neural Networks

Christian M.M. Frey, Yunpu Ma, Matthias Schubert
Institute for Informatics
Oettingenstr. 67
80538 Munich, Germany
Christian.Frey@lmu.de, {ma, schubert}@dbs.ifi.lmu.de

Abstract
A common issue in Graph Neural Networks (GNNs) is known as over-smoothing. By increasing the number of iterations within the message-passing of GNNs, the nodes’ representations of the input graph align with each other and become indiscernible. Recently, it has been shown that increasing a model’s complexity by integrating an attention mechanism yields more expressive architectures. This is majorly contributed to steering the nodes’ representations only towards nodes that are more informative than others. Transformer models in combination with GNNs result in architectures including Graph Transformer Layers (GTL), where layers are entirely based on the attention operation. However, the calculation of a node’s representation is still restricted to the computational working flow of a GNN. In our work, we relax the GNN architecture by means of implementing a routing heuristic. Specifically, the nodes’ representations are routed to dedicated experts. Each expert calculates the representations according to their respective GNN workflow. The definitions of distinguishable GNNs result from k-localized views starting from the central node. We call this procedure Graph Shell Attention (SEA), where experts process different subgraphs in a transformer-motivated fashion. Intuitively, by increasing the number of experts, the models gain in expressiveness such that a node’s representation is solely based on nodes that are located within the receptive field of an expert. We evaluate our architecture on various benchmark datasets showing competitive results compared to state-of-the-art models.

Introduction
The modeling flow of Graph Neural Networks (GNNs) has been proven to be a convenient tool in a variety of real-world applications building on top of graph data (Wu et al. 2021). These range from predictions in social networks over property predictions in molecular graph structures to content recommendations in online platforms. From a machine learning perspective, we can categorize them into various theoretical problems that are known as node classification, graph classification/regression - encompassing binary decisions or modeling a continuous-valued function -, and relation prediction. In our work, we propose a novel framework and show its applicability on graph-level classification and regression, as well as on node-level classification tasks.

The high-level intuition behind GNNs is that by increasing the number of iterations $k = 1, \ldots, K$, a node’s representation processes, contains, and therefore relies more and more on its $k$-hop neighborhood. However, a well-known issue with the vanilla GNN architecture refers to a problem called over-smoothing (Hoory, Linial, and Wigderson 2006; Xu et al. 2018a). Suppose we are given a GNN-motivated model, the information flow between two nodes $u, v \in \mathcal{V}$, where $\mathcal{V}$ denotes a set of nodes, is proportional to the reachability of node $v$ on a $k$-step random walk starting from $u$. Hence, by increasing the layers within the GNN architecture, the information flow of every node approaches the stationary distribution of random walks over the graph. As a consequence, the localized information flow is getting lost. On graph data that follow strong connectivity, it takes $k = O(\log |\mathcal{V}|)$ steps for a random walk starting from an arbitrary node to converge to an (almost) uniform distribution. Consequently, increasing the number of iterations within the GNN message-passing results in representations for all the nodes in the input graph that align and become indiscernible.

One strategy for increasing a GNN’s expressiveness is by adding an attention mechanism into the architecture. Recently, it has been shown how the Transformer model (Vaswani et al. 2017) can be applied on graph data (Dwivedi and Bresson 2021) yielding competitive results to state-of-the-art models. Generally, multi-headed attention shows varying results whenever we have prior knowledge to indicate that some neighbors might be more informative than others.

Our framework further improves the representational capacity by adding an expert motivated heuristic into the GNN architecture. More specifically, to compute a node’s representation, a routing module first decides upon an expert that is responsible for a node’s computation. The experts differ in how their $k$-hop localized neighborhood is processed and capture various depths of GNNs. We refer to the different substructures that individual experts process by Graph Shells. As each expert attends to a specific subgraph of the input graph, we introduce the concept of Graph Shell Attention (SEA). Hence, whereas a vanilla GNN lacks in over-smoothing the representations for all nodes, we introduce additional degrees of freedom in our architecture to simultaneously capture short- and long-term dependencies being processed by respective experts.

In summary, our contributions are as follows:

- Integration of expert-routing mechanism into Trans-
former motivated models applied on graph data;
• Novel Graph Shell Attention (SEA) models relaxing the vanilla GNN definition;

Related Work
In recent years, the AI community proposed various forms of (self-)attention in numerous domains. Attention itself refers to a mechanism in neural networks where a model learns to make predictions by selectively attending to a given set of data. The success of these models dates back to Vaswani et al. (Vaswani et al. 2017) by introducing the Transformer model. It relies on scaled dot-product attention, i.e., given a query matrix \( Q \), a key matrix \( K \), and a value matrix \( V \), the output is a weighted sum of the value vectors, where the dot-product of the query with corresponding keys determines the weight that is assigned to each value.

Transformer architectures have also been successfully applied to graph data. A thorough work by Dwivedi et al. (Dwivedi and Bresson 2021) evaluates transformer-based GNNs. They conclude that the attention mechanism in Transformers applied on graph data should only aggregate the information from the local neighborhood, ensuring graph sparsity. As in Natural Language Processing (NLP), where a positional encoding is applied, they propose to use Laplacian eigenvectors as the positional encodings for further improvements. In their results, they outperform baseline GNNs on the graph representation task. A similar work (Kreuzer et al., 2021) proposes a full Laplacian spectrum to learn the position of each node within a graph. Yun et al. (Yun et al. 2019) proposed Graph Transformer Networks (GTN) that is capable of learning on heterogeneous graphs. The target is to transform a given heterogeneous input graph into a meta-path-based graph and apply a convolution operation afterward. Hence, the focus of their attention framework is on interpreting generated meta-paths. Another transformer-based architecture that has been introduced by Hu et al. (Hu et al. 2020b) is Heterogeneous Graph Transformer (HTG). Notably, their architecture can capture graph dynamics w.r.t the central node and the message-passing nodes. By including the temporal information, Zhou et al. (Zhou et al. 2020) built a transformer-based generative model for generating temporal graphs by directly learning from the dynamic information in networks. The work of Nguyen et al. (Nguyen, Nguyen, and Phung 2019) proposes another idea for positional encoding. The authors of this work introduced a graph transformer for arbitrary homogeneous graphs with a coordinate embedding-based positional encoding scheme.

In (Ying et al. 2021), the authors introduced a transformer motivated architecture where various encodings are aggregated to compute the hidden representations. They propose graph structural encodings subsuming a spatial encoding, an edge encoding, and a centrality encoding.

Furthermore, a work exploring the effectiveness of large-scale pre-trained GNN models is proposed by the GROVER model (Rong et al. 2020). The authors include an additional GNN applied in the attention sublayer to produce vectors for \( Q \), \( K \), and \( V \). Moreover, they apply single long-range residual connections and two branches of feedforward networks to produce node and edge representations separately. In a self-supervised fashion, they first pre-train their model on 10 million unlabeled molecules before using the resulting node representations in downstream tasks.

Typically, all the models are built in a way such that the same parameters are used for all inputs. To gain more expressiveness, the motivation of the mixture of experts (MoE) heuristic (Shazeer et al., 2017) is to apply different parameters w.r.t to the input data. Recently, Google proposed Switch Transformer (Fedus, Zoph, and Shazeer 2021), enbling training above a trillion parameter networks but keeping the computational cost in the inference step constant. In our work, we will show how we can apply MoE motivated heuristics in the scope of GNN models.

Preliminaries
In this section, we provide definitions and recap on the general message-passing paradigm combined with Graph Transformer Layers (GTLs) (Dwivedi and Bresson 2021).

Notation
Let \( G = (V, E) \) be an undirected graph where \( V \) denotes a set of nodes and \( E \) denotes a set of edges connecting nodes. We define \( N_k(u) \) to be the \( k \)-hop neighborhood of a node \( u \in V \), i.e., \( N_k(u) = \{ v \in V : d_G(u, v) \leq k \} \), where \( d_G(u, v) \) denotes the hop-distance between \( u \) and \( v \) on \( G \). For \( N_k(u) \) we will simply write \( N(u) \) and omit the index \( k \). The induced subgraph by including the \( k \)-hop neighbors starting from node \( u \) is denoted by \( G^k_u \). Moreover, in the following we will use a real-valued representation vector \( h_u \in \mathbb{R}^d \) for a node \( u \), where \( d \) denotes the embedding dimensionality.

Graph Neural Networks
Given a graph \( G = (V, E) \) with node attributes \( X_V = \{X_u | u \in V \} \) and edge attributes \( X_E = \{X_{uv} | (u, v) \in E \} \), a GNN aims to learn an embedding vector \( h_u \) for each node \( u \in V \), and a vector \( h_e \) for the entire graph \( G \). For an L-layer GNN, a neighborhood aggregation scheme is performed to capture the L-hop information surrounding each node. The l-th layer of a GNN is formalized as follows:

\[
h^{l+1}_u = \text{UPDATE}(h^l_u, m^l_{N(u)}) \quad (1)
\]

\[
m^l_{N(u)} = \text{AGGREGATE}((\{h^l_v : v \in N(u)\}), \quad (2)
\]

where \( N(u) \) is the 1-hop neighborhood set of \( u \), \( h^{(l)}_u \) denotes the representation of node \( u \) at the l-th layer, and \( h^{(0)}_u \) is initialized as the node attribute \( X_u \). Since \( h_u \) summarizes the information of central node \( u \), it is also referred as patch embedding in the literature. A graph’s embedding \( h_G \) is derived by a permutation-invariant readout function:

\[
h_G = \text{READOUT}(\{h_u | u \in V\}) \quad (3)
\]

A common heuristic for the readout function is to choose a function \( \text{READOUT}() \in \{\text{mean}(), \text{sum}(), \text{max}()\} \).
Transformer

The vanilla Transformer architecture as proposed by Vaswani et al. (Vaswani et al. 2017) was originally introduced in the scope of Natural Language Processing (NLP) and consists of a composition of Transformer layers. Each Transformer layer has two parts: a self-attention module and a position-wise feedforward network (FFN). Let $H = [h_1, \ldots, h_n]^T \in \mathbb{R}^{n \times d}$ denote the input of the self-attention module where $d$ is the hidden dimension and $h_i \in \mathbb{R}^{1 \times d}$ is the hidden representation at position $i$ of an input sequence. The input $H$ is projected by three matrices $W_Q \in \mathbb{R}^{d \times d_Q}$, $W_K \in \mathbb{R}^{d \times d_K}$, and $W_V \in \mathbb{R}^{d \times d_V}$ to get the corresponding representation $Q, K, V$. The self-attention is calculated as:

$$Q = HW_Q \quad K = HW_K \quad V = HW_V,$$

$$A = \frac{QK^T}{\sqrt{d_k}}, \quad \text{Attn}(H) = \text{softmax}(A)V \quad (4)$$

Notably, in NLP as well as in computer vision tasks, the usage of transformer models were boosters behind a large number of state-of-the-art systems. Recently, the Transformer architecture has also been modified to be applicable to graph data (Dwivedi and Bresson 2021), which we will briefly recap in the next section.

Graph Transformer Layer

Generally, in NLP, the words in an input sequence can be represented as a fully connected graph that includes all connections between the input words. However, such an architecture does not leverage graph connectivity. Therefore, it can perform poorly whenever the graph topology is important and has not been encoded in the input data in any other way. These thoughts lead to the work (Dwivedi and Bresson 2021), where a general Graph Transformer Layer (GTL) has been introduced.

For the sake of completeness, we will recap on the most important equations. A layer update for layer $l$ within a GTL including edge features is defined as:

$$\hat{h}_{i}^{l+1} = O_h^{l}\left[ D^{\frac{1}{2}} \sum_{v \in \mathcal{N}_i} w_{iv}^{k,l} V^{k,l} h_{v}^{l}\right], \quad (5)$$

$$\hat{e}_{uv}^{l+1} = O_e^{l}\left[ \text{softmax}_{v}(\hat{w}_{uv}^{k,l})\right], \quad (6)$$

$$w_{uv}^{k,l} = \text{softmax}_{v}(\hat{w}_{uv}^{k,l}), \quad (7)$$

$$\hat{w}_{uv}^{k,l} = \left(\frac{Q^{k,l}h_{u}^{l}, K^{k,l}h_{v}^{l}}{\sqrt{d_k}}\right) \cdot E^{k,l} e_{uv}^{l}, \quad (8)$$

where $Q^{k,l}, K^{k,l}, V^{k,l}, E^{k,l} \in \mathbb{R}^{d_k \times d}$, and $O_h^{l}, O_e^{l} \in \mathbb{R}^{d \times d}$. The operator $\| \|$ denotes the concatenation of attention heads $k = 1, \ldots, H$. Subsequently, the outputs $\hat{h}_{i}^{l+1}$ and $\hat{e}_{uv}^{l+1}$ are passed to feedforward networks and succeed by residual connections and normalization layers as in the vanilla Transformer architecture (Vaswani et al. 2017). The full architecture of GTLs is illustrated in figure [1].

Positional Encoding. Using the Laplacian eigenvectors as the positional encoding for nodes within a graph is a common heuristic (Dwivedi and Bresson 2021). From spectral analysis, the eigenvectors are computed by the factorization of the Laplacian matrix:

$$\Delta = I - D^{-1/2} AD^{-1/2} = U^T \Lambda U, \quad (9)$$

where $A$ denotes the $n \times n$ adjacency matrix, $D$ denotes the degree matrix. The decomposition yields $\Lambda, U$ corresponding to the eigenvalues and eigenvectors. For the positional encoding, we use the $k$ smallest eigenvectors of a node. In our models, we also apply the Laplacian Positional Encoding (LPE).

Methodology

In this section, we introduce our Graph Shell Attention (SEA) architecture for graph data. SEA builds on top of the message-passing paradigm of Graph Neural Networks (GNNs) and integrates an expert heuristic. Over-smoothing in GNNs is a well-known issue (Xu et al. 2018a) and exacerbates the problem when we build deeper GNN models which generate similar representations for all nodes in the graph. Applying the same number of iterations for each node inhibits the expressiveness of short- and long-term dependencies simultaneously. Therefore, our architecture loosens up the strict workflow that is applied for each node equally. We gain expressiveness by routing each node representation towards dedicated experts, which only process nodes in their $k$-localized receptive field.

Graph Shells

In our approach, we exploit the graph transformer architecture, including Graph Transformer Layers (GTLs) (Dwivedi and Bresson 2021) and extend it by a set of experts. A routing layer decides upon which expert is most relevant for the

\[
\begin{align*}
Q &= HW_Q \\
K &= HW_K \\
V &= HW_V,
\end{align*}
\]

\[
A = \frac{QK^T}{\sqrt{d_k}}, \quad \text{Attn}(H) = \text{softmax}(A)V
\]

\[
\hat{h}_{i}^{l+1} = O_h^{l}\left[ D^{\frac{1}{2}} \sum_{v \in \mathcal{N}_i} w_{iv}^{k,l} V^{k,l} h_{v}^{l}\right], \quad (5)
\]

\[
\hat{e}_{uv}^{l+1} = O_e^{l}\left[ \text{softmax}_{v}(\hat{w}_{uv}^{k,l})\right], \quad (6)
\]

\[
w_{uv}^{k,l} = \text{softmax}_{v}(\hat{w}_{uv}^{k,l}), \quad (7)
\]

\[
\hat{w}_{uv}^{k,l} = \left(\frac{Q^{k,l}h_{u}^{l}, K^{k,l}h_{v}^{l}}{\sqrt{d_k}}\right) \cdot E^{k,l} e_{uv}^{l}, \quad (8)
\]

\[
\Delta = I - D^{-1/2} AD^{-1/2} = U^T \Lambda U, \quad (9)
\]

\[
\begin{align*}
Q &= HW_Q \\
K &= HW_K \\
V &= HW_V,
\end{align*}
\]

\[
A = \frac{QK^T}{\sqrt{d_k}}, \quad \text{Attn}(H) = \text{softmax}(A)V
\]

\[
\hat{h}_{i}^{l+1} = O_h^{l}\left[ D^{\frac{1}{2}} \sum_{v \in \mathcal{N}_i} w_{iv}^{k,l} V^{k,l} h_{v}^{l}\right], \quad (5)
\]

\[
\hat{e}_{uv}^{l+1} = O_e^{l}\left[ \text{softmax}_{v}(\hat{w}_{uv}^{k,l})\right], \quad (6)
\]

\[
w_{uv}^{k,l} = \text{softmax}_{v}(\hat{w}_{uv}^{k,l}), \quad (7)
\]

\[
\hat{w}_{uv}^{k,l} = \left(\frac{Q^{k,l}h_{u}^{l}, K^{k,l}h_{v}^{l}}{\sqrt{d_k}}\right) \cdot E^{k,l} e_{uv}^{l}, \quad (8)
\]

\[
\Delta = I - D^{-1/2} AD^{-1/2} = U^T \Lambda U, \quad (9)
\]
computation of a node’s representation. Intuitively, the expert’s computation for a node representation differs in how \(k\)-hop neighbors are stored and processed within GTLs.

Generally, starting from a central node, Graph Shells refer to subgraphs that include only nodes that have at maximum a \(k\)-hop distance (\(k\)-neighborhood). More formally, the \(i\)-th expert comprises the information given in the \(i\)-th neighborhood \(N_i(u) = \{v \in V : d_G(u, v) \leq i\}\), where \(u \in V\) denotes the central node. We refer to the subgraph \(G^*_i\) as the expert’s receptive field. Notably, increasing the number of iterations within GNNs correlates with the number of experts being used.

In the following, we introduce three variants on how experts process graph shells.

**SEA-GNN**

For the definition of the first graph shell model, we exploit the GNN architecture. Hence, the shells described by each expert are given by construction. From the formal definition expressed in eqs. 1 and 2, the information for the \(i\)-th expert is defined by the \(i\)-th iteration of the GNN. For \(N\) experts, we set the maximal number of hops to be \(L = N\). Figure 2a illustrates this graph shell model. From left to right, the information of nodes being reachable by more hops is processed. Experts storing information after few hops refer to short-term dependencies, whereas experts processing more hops yields information of long-term dependencies.

**SEA-AGGREGATED**

For the computation of the hidden representation \(h^{l+1}_{i+1}\) for node \(u\) on layer \(l + 1\), the second model employs an aggregated value from the previous iteration. According to the message-passing paradigm of a vanilla GNN, the aggregation function of eq. 2 considers the aggregated value \(m_{N_1(u)}^l\) to all its 1-hop neighbors. For a node \(v \in N_1(u)\), the values received by \(v\) are processed according to another aggregation function which can be chosen among \(\text{AGGREGATE}_\mu \in \{\text{mean}(\cdot), \text{sum}(\cdot), \text{max}(\cdot)\}\). Formally:

\[
\begin{align*}
  h_{u}^{l+1} &= \text{AGGREGATE}_\mu (m_{N_1(u)}^l; v \in N(u)) \\
  m_{N_k(u)}^l &= \text{AGGREGATE}_\mu (\{h_{v}^{l} : v \in N_k(u)\}),
\end{align*}
\]

Figure 2b illustrates this graph shell model. In the first iteration, there are no proceeding layers. Hence, the first expert processes in the same way as the first model. The representations for the second expert are computed by taking the aggregated values of the first shell into account. Sending values to neighboring nodes is illustrated by a full-colored shell.

**SEA-K-HOP**

For this model we relax the \(\text{AGGREGATE}\) function defined in eq. 2. Given a graph \(G\), we also consider \(k\)-hop linkages in the graph connecting a node \(u\) with all entities having a maximum distance of \(d_G(u, v) = k\). The relaxations of eqs. 1 and 2 can then be written as:

\[
\begin{align*}
  h_{u}^{l+1} &= \text{UPDATE}_\mu (h_{u}^{l}, m_{N_k(u)}^l) \\
  m_{N_k(u)}^l &= \text{AGGREGATE}_\mu (\{h_{v}^{l} : v \in N_k(u)\}),
\end{align*}
\]

where \(N_k(u)\) denotes the \(k\)-hop neighborhood set. This approach allows for processing each \(N_1(u), \ldots, N_k(u)\) with their own sub-modules, i.e., for each \(k\)-hop neighbors we use respective feedforward networks to compute \(Q, K, V\) in GTLs. Notably, this definition can be interpreted as a generalization of the vanilla GNN architecture, which is given by setting \(k = 1\). Figure 2c shows the \(k\)-hop graph shell model with \(k = 2\). From the first iteration on, we take the \(k\)-hop neighbors into account to calculate a node’s representation. Dotted edges illustrate the additional information flow.

**SEA: Shell Attention**

By endowing our models with experts referring to various graph shells, we gain expressiveness. A routing module decides to which graph shell the attention is steered. We apply a single expert strategy (Fedus, Zoph, and Shazeer 2021).

Originally introduced for language modeling and machine translation, Shazeer et al. (Shazeer et al. 2017) proposed a Mixture-of-Experts (MoE) layer. The general idea relies on a routing mechanism for token representations \(x\) to determine the best expert from a set \(\{E_i(x)\}_{i=1}^{N}\) of \(N\) experts. The router module consists of a single linear transformation whose output is normalized via softmaxing over the available \(N\) experts. Hence, the probability of choosing the \(i\)-th expert for node \(u\) is given as:

\[
p_{i}(u) = \frac{\exp(r(u)i)}{\sum_{j} \exp(r(u)j)}, \quad r(u) = h_{u}^{T}W_{r}
\]
The idea is to select the expert \( E_i \) representative for a node’s representation, i.e., where \( \arg \max \) linear transformation:

\[
E_i = \arg \max_{i \in [1, \ldots, N]} (R \cdot h_{u,i} + b_i) \quad \text{for computing } \hat{y}_{\text{LPE}}, \forall u \in V \;
\]

where \( r(\cdot) \) denotes the routing operation for a node \( u \) with \( W_r \in \mathbb{R}^{d \times N} \) being the routing’s learnable weight matrix. The idea is to select the expert \( E_i(\cdot) \) that is the most representative for a node’s representation, i.e., where \( i^* = \arg \max_{i \in [1, \ldots, N]} p_i(u) \). A node’s representation calculated by the chosen expert is then used as input for an expert’s individual linear transformation:

\[
h_{u,i^*} = E_{i^*}(u)^T W_{i^*} + b_{i^*},
\]

where \( W_{i^*} \in \mathbb{R}^{d \times d} \) denotes the weight matrix of expert \( E_{i^*}(\cdot) \), \( b_{i^*} \) denotes the bias term. The node’s representation according to expert \( E_{i^*}(\cdot) \), is denoted by \( h_{u,i^*} \). The architecture on how the routing mechanism is integrated into our model is shown in figure 3.

**Evaluation**

**Experimental Setting**

**Datasets**

**ZINC** (Irwin et al. 2012) is one of the most popular real-world molecular dataset consisting of 250K graphs. A subset consisting of 10K train, 1K validation, and 1K test graphs is used in the literature as benchmark (Dwivedi et al. 2020). The task is to regress a molecular property known as the constraint solubility. Each molecular graph, the node features are types of heavy atoms, and edge features are types of bonds between them.

We also evaluate our models on **ogbg-molhiv** (Hu et al. 2020a). Each graph within the dataset represents a molecule, where nodes are atoms and edges are chemical bonds. The task is to predict the target molecular property that is cast as binary label, i.e., whether a molecule inhibits HIV replication or not. It is known that this dataset suffers from a de-correlation between validation and test set performance.

A benchmark dataset generated by the **Stochastic Block Model (SBM)** (Abbe 2018) is **PATTERN**. The graphs within this dataset do not have explicit edge features. The task is to classify the nodes into 2 communities. The size of this dataset encompasses 14K graphs.

The benchmark datasets are summarized in table 1.

**Implementation Details**

Our implementation builds upon PyTorch (Paszke et al. 2019), DGL (Wang et al. 2019), and OGB (Hu et al. 2020a). We trained our models on a single GPU, an NVIDIA GeForce RTX 2080 Ti.

**Model Configuration**

We use the Adam optimizer (Kingma and Ba 2015) with an initial learning rate selected in \( \{10^{-3}, 10^{-5}\} \). We apply the same learning rate decay strategy for all models that half the learning rate if the validation loss does not improve over a fixed number of 5 epochs.

We tune the pairing \((\#\text{heads}, \text{hidden dimension}) \in \{(4, 32), (8, 64), (8, 56)\}\) and use \textsc{readout} \( \in \{\text{sum}\} \) as function for inference on the whole graph information. \textsc{Batch Normalization} and \textsc{Layer Normalization} are disabled, whereas residual connections are activated per default. For dropout we tuned the value to be \( \in \{0.0, 0.01, 0.05, 0.07, 0.1\} \) and a weight decay \( \in \{0.5e-5\} \). For the number of graph shells, i.e., number of experts being used, we report values \( \in \{4, 6, 8, 10, 12\} \). As aggregation functions we use \textsc{aggregate} \( \in \{\text{sum}\} \) and \textsc{aggregate}_{k} \( \in \{\text{mean}\} \). For LPE, the 8 smallest eigenvectors are used.

**Prediction Tasks**

In the following series of experiments, we investigate the performance of the **Graph Shell Attention** mechanism on graph-level prediction tasks for the datasets ogbg-molhiv (Hu et al. 2020a) and ZINC (Irwin et al. 2012) and a node-level classification task on PATTERN (Abbe 2018). We use commonly used metrics for the prediction tasks, i.e., mean absolute error (MAE) for ZINC, the ROC-AUC score on ogbg-molhiv, and the accuracy on PATTERN.

**Competitors.** We evaluate our architectures against state-of-the-art GNN models achieving competitive results. Our report subsumes the vanilla GCN (Kipf and Welling 2017), GAT (Veličković et al. 2017) that includes additional attention heuristics, or more recent GNN architectures building on top of Transformer-enhanced models like SAN (Kreuzer et al. 2021) and Graphormer (Ying et al. 2021). Moreover, we include GIN (Xu et al. 2018a) that is more discriminative towards graph structures compared to GCN (Kipf and Welling 2017), GraphSage (Hamilton, Ying, and Leskovec 2017), and DGN (Beani et al. 2020) being more discriminative than standard GNNs w.r.t the Weisfeiler-Lehman 1-WL test.

**Results.** Tables 2, 3, and 4 summarize the performances of our **SEA** models compared to baselines on ZINC, ogbg-molhiv, and PATTERN. **Vanilla GTL** shows the results of our implementation of the GNN model including Graph Transformer Layers (Dwivedi and Bresson 2021). **SEA-2-HOP** includes the 2-hop connection within the input graph, whereas **SEA-2-HOP-AUG** process the input data the same way as the 2-HOP heuristic, but uses additional feedforward networks for computing \( Q, K, V \) values for the 2-hop neighbors.
For PATTERN, we observe the best result using the **SEA-2-HOP** model, beating all other competitors. On the other hand, distributing an aggregated value to neighboring nodes according to **SEA-AGGREGATED** yields a too coarse view for graphs following the SBM and loses local graph structure. In the sense of **Green AI** (Schwartz et al. 2020) that focuses on reducing the computational cost to encourage a reduction in the resources spent, our architecture reaches state-of-the-art performance on ogbg-molhiv while reducing the number of parameters being trained. Comparing **SEA-AGGREGATED** to the best result reported for **Graphormer** (Ying et al. 2021), our model economizes on **99.71%** of the number of parameters while still reaching competitive results.

The results on ZINC enforce the argument of using individual experts compared to vanilla GTLs, where the best result is reported for **SEA-2-HOP**.
The number of parameters stays the same by increasing no extra linear layers are used for each argumentation is on a full graph setting. In our model, we send information over non-existent edges, i.e., the parameter \( k \) in their respective receptive field. Therefore, we gain expressiveness by capturing varying short- and long-term dependencies expressed by individual experts.

For that, our model exploits the idea of a routing layer, which enables to steer the nodes’ representations towards the individual expressiveness of dedicated experts. As experts process different subgraphs starting from a central node, we introduce the terminology of Graph Shell Attention (SEA), where experts solely process nodes that are in their respective receptive field. Therefore, we gain expressiveness by capturing varying short- and long-term dependencies expressed by individual experts.

In a thorough experimental study, we show on real-world benchmark datasets that the gained expressiveness yields competitive performance compared to state-of-the-art results while reducing the number of parameters. Additionally, we report a series of experiments that stress the number of graph shells that are taken into account.

In the future, we aim to work on more novel implementations and applications enhanced with the graph shell attention mechanism, e.g., where different hyperparameters are used for different graph shells.

We introduced the theoretical foundation of integrating an expert heuristic within transformer-based graph neural networks (GNNs). This opens a fruitful direction for future works that go beyond successive neighborhood aggregation (message-passing) to develop even more powerful architectures in graph learning.

We provide an engineered solution that allows selecting the most representative experts for nodes in the input graph. For that, our model exploits the idea of a routing layer, which enables to steer the nodes’ representations towards the individual expressiveness of dedicated experts.

As experts process different subgraphs starting from a central node, we introduce the terminology of Graph Shell Attention (SEA), where experts solely process nodes that are in their respective receptive field. Therefore, we gain expressiveness by capturing varying short- and long-term dependencies expressed by individual experts.

In a thorough experimental study, we show on real-world benchmark datasets that the gained expressiveness yields competitive performance compared to state-of-the-art results while reducing the number of parameters. Additionally, we report a series of experiments that stress the number of graph shells that are taken into account.

In the future, we aim to work on more novel implementations and applications enhanced with the graph shell attention mechanism, e.g., where different hyperparameters are used for different graph shells.

### References

Abbe, E. 2018. Community Detection and Stochastic Block Models. *Found. Trends Commun. Inf. Theory*, 14(1–2): 1–162.

Beaini, D.; Passaro, S.; Létourneau, V.; Hamilton, W. L.;
Corso, G.; and Liò, P. 2020. Directional Graph Networks. CoRR, abs/2010.02863.

Dwivedi, V. P.; and Bresson, X. 2021. A Generalization of Transformer Networks to Graphs. arXiv:2012.09699.

Dwivedi, V. P.; Joshi, C. K.; Laurent, T.; Bengio, Y.; and Bresson, X. 2020. Benchmarking Graph Neural Networks. CoRR, abs/2003.00982.

Fedus, W.; Zoph, B.; and Shazeer, N. 2021. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. CoRR, abs/2101.03961.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., Advances in Neural Information Processing Systems, volume 30. Curran Associates, Inc.

Hoory, S.; Linial, N.; and Wigderson, A. 2006. Expander Graphs and Their Applications. Bull. Amer. Math. Soc., 43(04): 439–562.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020a. Open Graph Benchmark: Datasets for Machine Learning on Graphs. arXiv preprint arXiv:2005.00687.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020b. Heterogeneous Graph Transformer. In Proceedings of The Web Conference 2020, WWW ’20, 2704–2710. New York, NY, USA: Association for Computing Machinery. ISBN 9781450370233.

Irwin, J. J.; Sterling, T.; Mysinger, M. M.; Bolstad, E. S.; and Coleman, R. G. 2012. ZINC: A Free Tool to Discover Chemistry for Biology. Journal of chemical information and modeling, 52(7): 1757–1768.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR).

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In Proceedings of the 5th International Conference on Learning Representations, ICLR ’17.

Kreuzer, D.; Beaini, D.; Hamilton, W. L.; Létourneau, V.; and Tossou, P. 2021. Rethinking Graph Transformers with Spectral Attention. arXiv:2106.03893.

Nguyen, D. Q.; Nguyen, T. D.; and Phung, D. 2019. Universal Self-Attention Network for Graph Classification. arXiv preprint arXiv:1909.11855.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d’Alché-Buc, F.; Fox, E.; and Garnett, R., eds., Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc.

Rong, Y.; Bian, Y.; Xu, T.; Xie, W.; WEI, Y.; Huang, W.; and Huang, J. 2020. Self-Supervised Graph Transformer on Large-Scale Molecular Data. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., Advances in Neural Information Processing Systems, volume 33, 12559–12571. Curran Associates, Inc.

Schwartz, R.; Dodge, J.; Smith, N. A.; and Etzioni, O. 2020. Green AI. Commun. ACM, 63(12): 54–63.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; and Dean, J. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv preprint arXiv:1701.06538.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, u.; and Polosukhin, I. 2017. Attention is All You Need. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17, 6000–6010. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781510860964.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2017. Graph Attention Networks. 6th International Conference on Learning Representations.

Wang, M.; Zheng, D.; Ye, Z.; Gan, Q.; Li, M.; Song, X.; Zhou, J.; Ma, C.; Yu, L.; Gai, Y.; Xiao, T.; He, T.; Karypis, G.; Li, J.; and Zhang, Z. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. arXiv preprint arXiv:1909.01315.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2021. A Comprehensive Survey on Graph Neural Networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1): 4–24.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018a. How Powerful are Graph Neural Networks? CoRR, abs/1810.00826.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018b. How Powerful are Graph Neural Networks? CoRR, abs/1810.00826.

Ying, C.; Cai, T.; Luo, S.; Zheng, S.; Ke, G.; He, D.; Shen, Y.; and Liu, T.-Y. 2021. Do Transformers Really Perform Bad for Graph Representation? arXiv preprint arXiv:2106.05234.

Yun, S.; Jeong, M.; Kim, R.; Kang, J.; and Kim, H. J. 2019. Graph Transformer Networks. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d’Alché-Buc, F.; Fox, E.; and Garnett, R., eds., Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc.

Zhou, D.; Zheng, L.; Han, J.; and He, J. 2020. A Data-Driven Graph Generative Model for Temporal Interaction Networks, 401–411. New York, NY, USA: Association for Computing Machinery. ISBN 9781450379984.

Supplementary Material

In the following, we provide additional information about the hyperparameters being used in our experiments. We also discuss the distribution of the experts being selected to learn the node representations of the input graph.
Notes on Hyperparameters

Table 7 summarizes the hyperparameter settings for the models SEA-GNN, SEA-AGGREGATED, SEA-2-HOP, and SEA-2-HOP-AUG being used individually for the datasets ZINC, ogbg-molhiv and PATTERN. Remarkably, in contrast to comparable architectures (Dwivedi and Bresson 2021; Kreuzer et al. 2021), we could not observe improvements with normalization techniques. Generally, batch normalization during training calculates the inputs’ mean and standard deviation to a layer per mini-batch and uses these statistics to perform the standardization. Note that in our architecture, each expert learns different statistics w.r.t to the information given by the graph shells they are processing, i.e., nodes within their respective fields. Hence, standardization techniques applied on mini-batches smooth out the peculiarities captured by the experts. Consequently, we disabled normalization techniques and just kept the residual connections enabled in the computational flow of graph transformer layers (GTLs). Smaller batch sizes for PATTERN are applied due to a limited 11 GB GDDR6 memory used in NVIDIA’s GeForce RTX 2080 Ti, which we used in our experiments. Notably, we also observed that disabling biases for feedforward networks yield better results. However, increasing the number of hops $k \geq 3$ within the SEA-K-HOP model, learning also biases result in better performances. In contrast to experiments shown in the approach proposed by Kreuzer et al. (Kreuzer et al. 2021), we also get better results using Adam as optimizer instead of AdamW.

Distribution of Experts

We evaluate the distributions of the experts being chosen to compute the nodes’ representations in the following. We set the number of experts to 8. Figure 4 summarizes the relative frequencies of the experts being chosen on the datasets ZINC, ogbg-molhiv, and PATTERN. Generally, the performance of the shell attention heuristic degenerates whenever we observe expert collapsing. In the extreme case, just one expert expresses the mass of all nodes, and the capability to distribute learning nodes’ representations over several experts is not leveraged. To overcome expert collapsing, we can use a heuristic where in the early stages of the learning procedure, an additional epsilon parameter $\epsilon$ introduces randomness into the algorithm. Like a decaying greedy policy in Reinforcement Learning (RL), we choose a random expert with probability $\epsilon$ and choose the expert with the highest probability according to the routing layer with a probability of $1 - \epsilon$. The epsilon value slowly decays over time. This ensures that all experts’ expressiveness is being explored to find the best matching one w.r.t to a node $u$ and prevents getting stuck in a local optimum. The figure shows the distribution of experts that are relevant for the computation of the nodes’ representations. For illustrative purposes, values below 1% are omitted. Generally, nodes are more widely distributed over all available experts in the molecular datasets like ZINC and ogbg-molhiv for all models compared to PATTERN following a stochastic block model. Therefore, various experts are capable of capturing individual topological characteristics of molecules better than vanilla graph neural networks for which over-smoothing might potentially occur. We also observe that the mass is distributed to only a subset of the available experts for the PATTERN dataset. Hence, the specific number of iterations is more expressive for nodes within graph structures following SBM.
Figure 4: Distribution of 8 experts for models SEA-GNN, SEA-AGGREGATED, SEA-2-HOP, and SEA-2-HOP-AUG for datasets ZINC, ogbg-molhiv and PATTERN. Relative frequencies are shown for values $\geq 1\%$. Numbers attached to the slices refer to the respective experts.