SEA: Graph Shell Attention in Graph Neural Networks

Christian M.M. Frey$^{1,2[0000-0003-2458-6651]}$✉, Yunpu Ma$^{2[0000-0001-6112-8794]}$, and Matthias Schubert$^{2[0000-0002-6566-6343]}$

^{1}Christian-Albrecht University of Kiel, Kiel, Germany
`cfr@informatik.uni-kiel.de`
^{2}Ludwig Maximilian University of Munich, Munich, Germany
`{ma, schubert}@dbs.ifi.lmu.de`

Abstract. A common problem 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 and become indiscernible. The latest models employing attention mechanisms with *Graph Transformer Layers* (GTLs) are still restricted to the layer-wise computational workflow of a GNN that are not beyond preventing such effects. 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 **S**h**e**ll **A**ttention* (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 while drastically reducing the number of parameters compared to state-of-the-art models.

1 Introduction

Graph Neural Networks (GNNs) have been proven to be an important tool in a variety of real-world applications building on top of graph data [22]. 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 $l = 1, \ldots, L$, a node's representation contains, and therefore relies successively more on its k-hop neighborhood. However, a well-known issue with the vanilla GNN architecture refers to a problem called *over-smoothing* [23]. In simple words, the information flow in GNNs 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. By increasing the layers within the GNN architecture, the information flow of every node approaches the stationary distribution of random walks over the graph [7]. As a consequence, the localized information flow is getting lost, i.e., increasing the number of iterations within the message-passing of GNN results in representations for all the nodes in the input graph that align and become indiscernible [15]. One strategy for increasing a GNN's effectiveness is adding an attention mechanism. An adaption of the *Transformer* model [19] on graph data has been introduced as *Graph Transformer Layer* (GTL) [3]. Generally, multi-headed attention shows competitive 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 heuristic into the GTL 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 they capture individually various depths of GNNs/GTLs. We refer to different substructures that experts process as *Graph Shells*. As each expert attends to a specific subgraph of the input graph, we introduce the concept of *Graph* ***S****hell* ***A****ttention* (SEA). Hence, whereas a vanilla GNN might suffer from over-smoothing the nodes' representations, 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 into graph neural nets;
– Novel Graph Shell Attention (SEA) models capturing short- and long-term dependencies, simultaneously;
– Experiments showing a reduction in the number of model parameters compared to SOTA models;

2 Related Work

In recent years, the AI community proposed various forms of (self-)attention mechanisms 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 applying attention heuristics was further boosted by introducing the *Transformer* model [19]. 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. The work by Dwivedi et al. [3] evaluates transformer-based GNNs. They conclude that the attention mechanism in Transformers applied on graph data should only aggregate the information from local neighborhoods, 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 [13] proposes a full Laplacian spectrum to learn the position of each node within a graph. Yun et al. [25] proposed *Graph Transformer Networks* (GTN) that are 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 afterwards. 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. [9] is *Heterogeneous Graph Transformer* (HGT). Notably, their architecture can capture graph dynamics w.r.t the information flow in heterogeneous graphs. Specifically, they take the relative temporal positional encoding into account based on differences of temporal information given for the central node and the message-passing nodes. By including the temporal information, Zhou et al. [26] built a transformer-based generative model for generating temporal graphs by directly learning from the dynamic information in networks. The work of Ngyuen et al. [14] 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 [24], 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 [16]. The authors include an additional GNN operating 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 [18] is to apply different parameters w.r.t the input data. Recently, Google proposed *Switch Transformer* [5], enabling training above a trillion parameter networks but keeping the computational cost in the inference step constant. We provide an approach how a similar routing mechanism can be integrated in GNNs.

3 Preliminaries

3.1 Notation

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph where $\mathcal{V}$ denotes a set of nodes and $\mathcal{E}$ denotes a set of edges connecting nodes. We define $N_k(u)$ to be the k-hop neighborhood of a node $u \in \mathcal{V}$, i.e., $N_k(u) = \{v \in \mathcal{V} : d_{\mathcal{G}}(u, v) \leq k\}$, where $d_G(u, v)$ denotes the hop-distance between u and v on $\mathcal{G}$. For $N_1(u)$ 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 $\mathcal{G}_u^k$. 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.

3.2 Recap: Graph Transformer Layer

As formalized in [3], a *Graph Transformer Layer* (GTL) update for layer $l \in [1..L]$ including edge features is defined as:

$$\hat{h}_u^{l+1} = O_h^l \mathop{\Big|\Big|}_{i=1}^{H} \Big(\sum_{v \in N(u)} w_{uv}^{i,l} V^{i,l} h_v^l \Big) , \tag{1}$$

$$\hat{e}_{uv}^{l+1} = O_e^l \mathop{\Big|\Big|}_{i=1}^{H} (\hat{w}_{uv}^{i,l}), \text{where} , \tag{2}$$

$$w_{uv}^{i,l} = \text{softmax}_v(\hat{w}_{uv}^{i,l}) , \tag{3}$$

$$\hat{w}_{uv}^{i,l} = \Big(\frac{Q^{i,l} h_u^l \cdot K^{i,l} h_v^l}{\sqrt{d_i}} \Big) \cdot E^{i,l} e_{uv}^l , \tag{4}$$

where $Q^{i,l}, K^{i,l}, V^{i,l}, E^{i,l} \in \mathbb{R}^{d_i \times d}$, and $O_h^l, O_e^l \in \mathbb{R}^{d \times d}$. The operator $||$ denotes the concatenation of attention heads $i = 1, \ldots, H$. Subsequently, the outputs $\hat{h}_u^{l+1}$ and $\hat{e}_{uv}^{l+1}$ are passed to feedforward networks and succeeded by residual connections and normalization layers yielding the representations h_u^{l+1} and e_{uv}^{l+1}.

A graph's embedding $h_{\mathcal{G}}$ is derived by a permutation-invariant readout function w.r.t. the nodes in $\mathcal{G}$:

$$h_{\mathcal{G}} = readout(\{h_u | u \in \mathcal{V}\}) \tag{5}$$

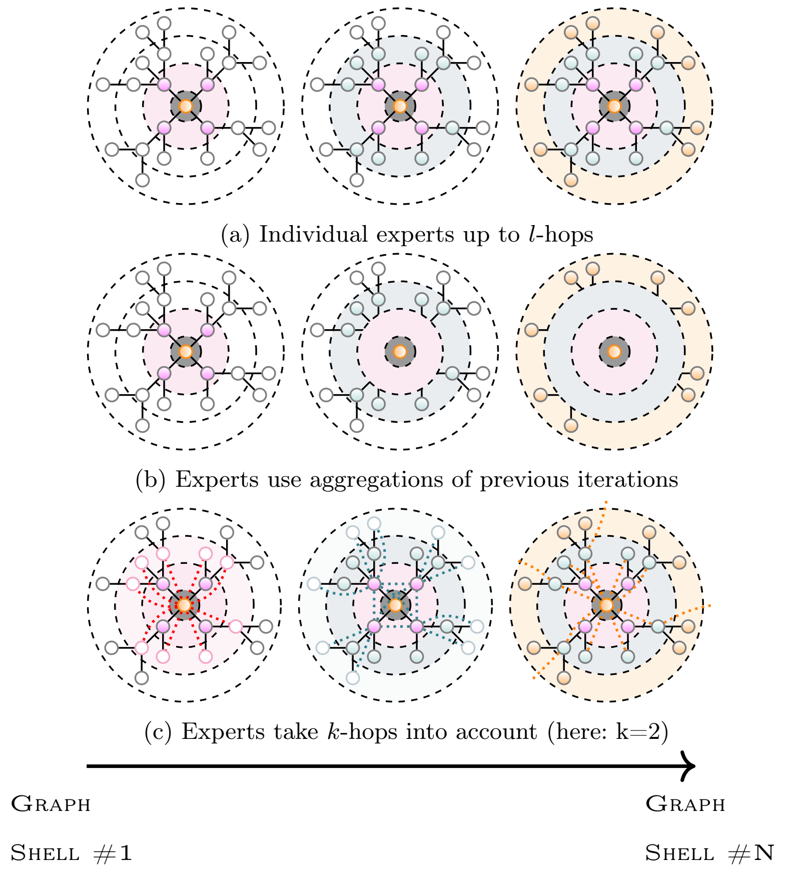
A common heuristic for the readout function is to choose a function $\textsc{readout}(\cdot) \in \{mean(\cdot), sum(\cdot), max(\cdot)\}$.

4 Methodology

In this section, we introduce our *Graph **Sh**ell **A**ttention* (SEA) architecture for graph data. SEA builds on top of the message-passing paradigm of *Graph Neural Networks* (GNNs) while integrating an expert heuristic.

4.1 Graph Shells Models

In our approach, we implement *Graph Transformer Layers* (GTLs) [3] and extend our framework by a set of *experts*. A routing layer decides which expert is most relevant for computing a node's representation. An expert's calculation for



(a) Individual experts up to l-hops

(b) Experts use aggregations of previous iterations

(c) Experts take k-hops into account (here: k=2)

Fig. 1: Three variants of SEA models; for each model, the respective fields of 3 experts are shown from left to right.

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). Formally, the i-th expert comprises the information given in the i-th neighborhood $N_i(u) = \{v \in \mathcal{V} : d_G(u, v) \leq i)\}$, where $u \in \mathcal{V}$ denotes the central node. We refer to the subgraph $\mathcal{G}_u^i$ as the expert's *receptive field*. Notably, increasing the number of iterations within GTLs/GNNs correlates with the number of experts being used. In the following, we introduce three variants on how experts process graph shells:

• **SEA-GTL.** The first graph shell model exploits the vanilla architecture of GTLs for which shells are defined by the standard graph neural net construction. For a maximal number of L iterations, we define a set $\{E_i(u)\}_{i=1}^{N}$ of $N = L$

experts. The embeddings after the l-th iteration are fed to the l-th expert, i.e., according to Eq. 1, the information of nodes in $\mathcal{G}_u^l$ for a central node u have been processed. Fig. 1a illustrates this model. From left to right, the information of nodes being reachable by more hops is processed. Experts processing information in early iterations refer to short-term dependencies, whereas experts processing more hops yield information of long-term dependencies.

• **SEA-AGGREGATED.** For the computation of the hidden representation h_u^{l+1} for node u on layer $l+1$, the second model employs an aggregated value from the preceding iteration. Following Eq. 1, the aggregation function (sum) in GLT considers all 1-hop neighbors $N_1(u)$. For *SEA*-AGGREGATED, we propagate the aggregated value back to all of u's 1-hop neighbors. For a node $v \in N_1(u)$, the values received by v are processed according to an aggregation function $\textsc{Agg} \in \{mean(\cdot), sum(\cdot), max(\cdot)\}$. Formally:

$$h_u^{l+1} = \textsc{Agg}^l(\{h_v^{l+1} : v \in N(u)\}) \tag{6}$$

Fig. 1b illustrates this graph shell model. In the first iteration, there are no preceding layers, hence, the first expert processes the information in the same way as in the first model. In succeeding iterations, the aggregated representations are first sent to neighboring nodes, which in turn process the incoming representations. These aggregated values define the input for the current iteration. Full-colored shells illustrate aggregated values from previous iterations.

• **SEA-K-HOP.** For this model we relax the aggregate function defined in Eq. 1. Given a graph $\mathcal{G}$, we also consider k-hop linkages in the graph connecting a node u with all entities having a maximum distance of $d_{\mathcal{G}}(u, v) = k$. The relaxation of Eq. 1 is formalized as:

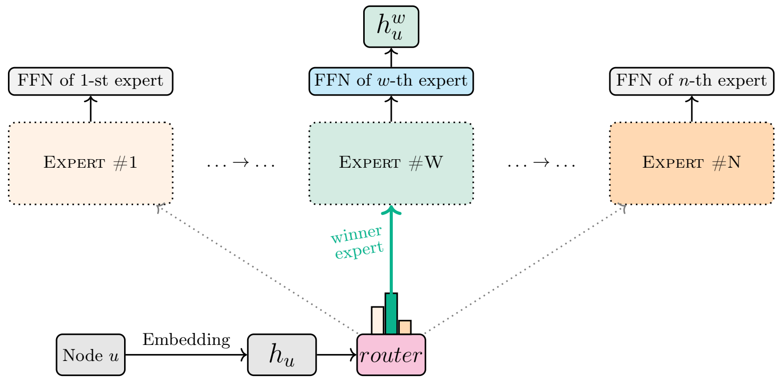


Fig. 2: Routing mechanism to N experts

$$\hat{h}_u^{l+1} = O_h^l \Big\Vert_{i=1}^{H} \Big(\sum_{v \in N_k(u)} \overset{\text{attention scores Eq. 3}}{w_{uv}^{i,l}} V^{i,l} \underset{\text{embds. of nodes in k-hop dist.}}{h_v^l} \Big) , \tag{7}$$

where $N_k(u)$ denotes the k-hop neighborhood set. This approach allows for processing each $N_1(u), \ldots, N_k(u)$ by own submodules, i.e., for each k-hop neighbors we use respective feedforward networks to compute Q, K, V in GTLs. Notably, Eq. 7 can be interpreted as a generalization of the vanilla architecture, which is given by setting $k = 1$. Fig. 1c shows the k-hop graph shell model with $k = 2$.

4.2 SEA: Routing Mechanism

By endowing our models with experts referring to various graph shells, we gain variable expressiveness for short- and long-term dependencies. Originally introduced for language modeling and machine translation, Shazeer et al. [18] proposed a Mixture-of-Experts (MoE) layer. A routing module decides to which expert the attention is steered. We use a *single expert* strategy [5].

The general idea relies on a routing mechanism for a node u's representation to determine the best expert from a set $\{E_i(u)\}_{i=1}^N$ of N experts processing graph shells as described in the previous section 4.1. The router module consists of a linear transformation whose output is normalized via softmaxing. The probability of choosing the i-th expert for node u is defined as:

$$p_i(u) = \frac{\exp(r(u)_i)}{\sum_j^N \exp(r(u)_j)} , \quad r(u) = h_u^T W_r + b_r , \tag{8}$$

where $r(\cdot)$ denotes the routing operation with $W_r \in \mathbb{R}^{d \times N}$ being the routing's learnable weight matrix, and b_r denotes a bias term. The idea is to select the winner expert $E_w(\cdot)$ that is the most representative for a node's representation, i.e, where $w = \underset{i=1,\dots,N}{\arg\max}\ p_i(u)$ [1]. A node's representation calculated by taking the winner's graph shells into account is then used as input for the expert's individual linear transformation:

$$h_u^w = E_w(u)^T W_w + b_w\,, \tag{9}$$

where $W_w \in \mathbb{R}^{d\times d}$ denotes the weight matrix of expert $E_w(\cdot)$, b_w denotes the bias term. The node's representation according to expert $E_w(\cdot)$, is denoted by h_u^w. Fig. 2 shows how the routing is integrated into our architecture.

4.3 Shells vs. Over-smoothing

Over-smoothing in GNNs is a well-known issue [23] and exacerbates the problem when we build deeper graph neural net models. Applying the same number of iterations for each node inhibits the simultaneous expressiveness of short- and long-term dependencies. We gain expressiveness by routing each node representation towards dedicated experts processing only nodes in their k-localized receptive field.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph. Following the proof scheme of [15], let $A = (\mathbb{1}_{(i,j)\in\mathcal{E}})_{i,j\in[N]:=\{1,\dots,N\}} \in \mathbb{R}^{N\times N}$ be the adjacency matrix and $D := \operatorname{diag}(\deg(i)_{i\in[N]}) \in \mathbb{R}^{N\times N}$ be the degree matrix of $\mathcal{G}$ where $\deg(i) := |\{j \in V \mid (i,j) \in \mathcal{E}\}|$ is the degree of node i. Let $\tilde{A} := A + I_N$, $\tilde{D} := D + I_N$ be the adjacent and the degree matrix of graph $\mathcal{G}$ augmented with self-loops, where I_N denotes the identity matrix of size N. The augmented normalized Laplacian of $\mathcal{G}$ is defined by $\tilde{\Delta} := I_N - \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ and set $P := I_N - \tilde{\Delta}$. Let $L, C \in \mathbb{N}_+$ be the layer and channel sizes, respectively. W.l.o.g, for weights $W_l \in \mathbb{R}^{C\times C}(l \in [L] := \{1,\dots,L\})$, we define a GCN associated with $\mathcal{G}$ by $f = f_L \circ \dots \circ f_1$ where $f_l : \mathbb{R}^{N\times C} \to \mathbb{R}^{N\times C}$ is defined by $f_l(X) = \sigma(PXW_l)$, where $\sigma(\cdot)$ denotes the ReLU activation function. For $M \leq N$, let U be a M-dimensional subspace of $\mathbb{R}^N$. Furthermore, we define a subspace $\mathcal{M}$ of $\mathbb{R}^{N\times C}$ by $\mathcal{M} = U \otimes \mathbb{R}^C = \{\sum_{m=1}^{M} e_m \otimes w_m \mid w_m \in \mathbb{R}^C\}$, where $(e_m)_{m\in[M]}$ is the orthonormal basis of U. For an input $X \in \mathbb{R}^{N\times C}$, the distance between X and $\mathcal{M}$ is denoted by $d_{\mathcal{M}} = \inf\{\|X - Y\|_F \mid Y \in \mathcal{M}\}$.

Considering $\mathcal{G}$ as M connected components, i.e. $V = V_1 \cup \dots \cup V_m$, where an indicator vector of the m-th connected component is denoted by $u_m = (\mathbb{1}_{\{n\in V_m\}})_{n\in[N]} \in R^N$. The authors of [15] investigated the asymptotic behavior of the output X^L of the GCN when $L \to \infty$:

Proposition 1. *Let $\lambda_1 \leq \dots \leq \lambda_N$ be the eigenvalue of P sorted in ascending order. Then, we have $-1 < \lambda_1, \lambda_{N-M} < 1$, and $\lambda_{N-M+1} = \dots = \lambda_N = 1$. In particular, we have $\lambda = \max_{n=1,\dots N-M}|\lambda_n| < 1$. Further, $e_m = \tilde{D}^{\frac{1}{2}}u_m$ for $m \in [M]$ are the basis of the eigenspace associated with the eigenvalue* 1.

[1] In DL libraries, the $\arg\max(\cdot)$ operation implicitly calls $\max(\dot{)}$ forwarding the maximum of the input. Hence, it is differentiable w.r.t to the values yielded by the max op., not to the indices

Table 1: Summary dataset statistics

Domain	Dataset	#Graphs	Task
Chemistry	ZINC	12K	Graph Regression
	OGBG-MOLHIV	41K	Graph Classification
Mathematical Modeling	PATTERN	14K	Node Classification

Let $s = \sup_{l \in \mathbb{N}_+} s_l$ with s_l denoting the maximum singular value of W_l, the major theorem and their implications for GCNs is stated as follows:

Theorem 1. *For any initial value $X^{(0)}$, the output of l-th layer $X^{(l)}$ satisfies $d_{\mathcal{M}}(X^{(l)}) \leq (s\lambda)^l d_{\mathcal{M}}(X^{(0)})$. In particular, $d_{\mathcal{M}}(X^{(l)})$ exponentially converges to 0 when $s\lambda < 1$.*

Proofs of Prop. 1 and Th. 1 are formulated in [15].

Intuitively, the representations X align subsequently with the subspace $\mathcal{M}$, where the distance between both converges to zero. Therefore, it can also be interpreted as information loss of graph neural nets in the limit of infinite layers.

The theoretical justification for the routing mechanism applied in our SEA models comes to light when we exploit the monotonous behavior of the exponential decay where the initial distance $d_{\mathcal{M}}(X^{(0)})$ is treated as a constant value. The architecture includes the experts in a cascading manner, where the routing mechanism allows to point to each of the $(d_{\mathcal{M}}(f_l(X)))_{l=1,\ldots,L}$, separately. From Th. 1, we get:

$$\begin{aligned} d_{\mathcal{M}}(X^{(L)}) &\leq (s\lambda)^L d_{\mathcal{M}}(X^{(0)}) \leq (s\lambda)^{L-1} d_{\mathcal{M}}(X^{(0)}) \\ &\leq \ldots \leq (s\lambda)^1 d_{\mathcal{M}}(X^{(0)}), \end{aligned}$$

where each inequality is supported by the output of the l-th expert, separately:

$$\begin{array}{lll} L\text{-th expert:} & d_{\mathcal{M}}(X^{(L)}) & \leq (s\lambda)^L d_{\mathcal{M}}(X^{(0)}) \\ L\text{-1-th expert:} & d_{\mathcal{M}}(X^{(L-1)}) & \leq (s\lambda)^{L-1} d_{\mathcal{M}}(X^{(0)}) \\ \ldots & \ldots & \leq \ldots \\ \text{1-st expert:} & d_{\mathcal{M}}(X^{(1)}) & \leq (s\lambda)^1 d_{\mathcal{M}}(X^{(0)}) \end{array}$$

Hence, our architecture does not suffer from overs-smoothing the same way as standard GNNs, as each captures a different distance $d_{\mathcal{M}}$ compared to using a GNN where a pre-defined number of layer updates is applied for all nodes equally and potentially leading to an over-smoothed representation.

5 Evaluation

5.1 Experimental Setting

Datasets.

ZINC [10] 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 [4].

We also evaluate our models on **ogbg-molhiv** [8]. Each graph within the dataset represents a molecule, where nodes are atoms and edges are chemical bonds.

A benchmark dataset generated by the *Stochastic Block Model* (SBM) [1] is **PATTERN**. The graphs within this dataset do not have explicit edge features. The benchmark datasets are summarized in Tab. 1.

Implementation Details.
Our implementation uses PyTorch, Deep Graph Library (DGL) [21], and OGB [8]. The models are trained on an NVIDIA GeForce RTX 2080 Ti. [2]

Model Configuration.
We use the Adam optimizer [11] with an initial learning rate $\in \{1\text{e-}3, 1\text{e-}4\}$. 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 $(\#\text{heads}, \text{hidden dimension}) \in \{(4, 32), (8, 56), (8, 64))\}$ and use READOUT $\in \{sum\}$ as function for inference on the whole graph information. *Batch Normalization* and *Layer Normalization* are disabled, whereas residual connections are activated per default in GTLs. For dropout, we tuned the value to be $\in \{0, 0.01, 0.05, 0.07, 0.1\}$ and a weight decay $\in \{5\text{e-}5, 5\text{e-}7\}$. For the number of graph shells, i.e, number of experts being used, we report values $\in \{4, 6, 8, 10, 12\}$. As aggregation function we use $\textsc{Agg} \in \{mean\}$ for Eq. 6. As laplacian encoding, the 8 smallest eigenvectors are used.

5.2 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 [8] and ZINC [10], and a node-level classification task on PATTERN [1]. We use commonly used metrics for the prediction tasks as they are used in [4], 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 [12], GAT [20] that includes additional attention heuristics, or more recent GNN architectures building on top of Transformer-enhanced models like SAN [13] and Graphormer [24]. Moreover, we include GIN [23] that is more discriminative towards graph structures compared to GCN [12], GraphSage [6], and DGN [2] being more discriminative than standard GNNs w.r.t the Weisfeiler-Lehman 1-WL test.

[2] Code: https://github.com/christianmaxmike/SEA-GNN

Table 2: Comparison to state-of-the-art; results are partially taken from [13, 4]; color coding (gold/silver/bronze)

	ZINC	
Model	**#params.**	**MAE**
GCN [12]	505K	0.367
GIN [23]	509K	0.526
GAT [20]	531K	0.384
SAN [13]	508K	**0.139**
Graphormer-SLIM [24]	489K	**0.122**
Vanilla GTL	83K	0.227
SEA-GTL	347K	0.212
SEA-AGGREGATED	112K	0.215
SEA-2-HOP	**430K**	**0.159**
SEA-2-HOP-AUG	709K	0.189

(a) ZINC [10]

	OGBG-MOLHIV	
Model	**#params.**	**%ROC-AUC**
GCN-GRAPHNORM [12]	526K	76.06
GIN-VN [23]	3.3M	77.80
DGN [2]	114K	79.05
Graphormer-FLAG [24]	47.0M	**80.51**
Vanilla GTL	386K	78.06
SEA-GTL	347K	79.53
SEA-AGGREGATED	**133K**	**80.18**
SEA-2-HOP	511K	**80.01**
SEA-2-HOP-AUG	594K	79.08

(b) ogbg-molhiv [8]

	PATTERN	
Model	**#params.**	**% ACC**
GCN [12]	500K	71.892
GIN [23]	100K	85.590
GAT [20]	526K	78.271
GraphSage [6]	101K	50.516
SAN [13]	454K	**86.581**
Vanilla GTL	82K	84.691
SEA-GTL	132K	85.006
SEA-AGGREGATED	69K	57.557
SEA-2-HOP	**48K**	**86.768**
SEA-2-HOP-AUG	152K	**86.673**

(c) PATTERN [1]

Results. Tables 2a, 2b, and 2c 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 [3]. *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* [17] that focuses on reducing the computational cost to encourage a reduction in resources spent, our architecture reaches state-of-the-art performance on ogbg-molhiv while drastically reducing the number of parameters being trained. Comparing *SEA*-AGGREGATED to the best result reported for *Graphormer* [24], our model economizes on $\mathbf{99.71}\%$ of the number of parameters while still reaching competitive results.

The results on ZINC enforces the argument of using individual experts compared to vanilla GTLs, where the best result is reported for *SEA*-2-HOP.

5.3 Number of Shells

Next, we examine the performance w.r.t the number of experts. Notably, increasing the number of experts correlated with the number of *Graph Shells* which are taken into account. Table 3 summarizes the results where all other hyperparameters are frozen, and we only have a variable size in the number of experts. We train each model for 500 epochs and report the best-observed metrics on the test datasets. We apply an early stopping heuristic, where we stop the learning procedure if we have not observed any improvements w.r.t the evaluation metrics or if the learning rate scheduler reaches a minimal value which we set to 10^{-6}. Each evaluation on the test data is conducted after 5 epochs, and the early stopping is effective after 10 consecutive evaluations on the test data with no improvements. First, note that increasing the number of experts also increases the model's parameters linearly. This is due to additional routings and linear layer being defined for each expert separately. Secondly, we report also the average running time in seconds $[s]$ on the training data for each epoch. By construction, the running time correlates with the number of parameters that have to be trained. The number of parameters differs from one dataset to another with the same settings due to a different number of nodes and edges within the datasets and slightly differs if biases are used or not. Note that we observe better results of *SEA*-AGGREGATED by decreasing the embedding size from 64 to 32, which also applies for the PATTERN dataset in general. The increase of parameters of the augmented 2-hop architecture *SEA*-2-HOP-AUG is due to the additional feedforward layers being used for the k-hop neighbors to compute the inputs Q, K, V in the graph transformer layer. Notably, we also observe that similar settings apply for datasets where the structure is an important feature of the graph,

Table 3: Influence of the number of experts applied on various SEA models; best configurations are highlighted in green

		ZINC			OGBG-MOLHIV			PATTERN		
Model	**#experts**	**#params**	**MAE**	**time/ epoch**	**#params**	**%ROC-AUC**	**time/ epoch**	**#params**	**% ACC**	**time/ epoch**
SEA-GTL	4	183K	0.385	13.60	182K	79.24	49.21	48K	78.975	58.14
	6	266K	0.368	20.93	263K	78.24	68.67	69K	82.117	82.46
	8	349K	**0.212**	26.24	345K	**79.53**	84.35	90K	82.983	108.41
	10	433K	0.264	31.63	428K	79.35	107.11	111K	84.041	133.73
	12	516K	0.249	38.26	511K	79.18	122.99	132K	**85.006**	168.47
SEA-AGGREGATED	4	49K	0.257	31.24	48K	77.87	60.98	48K	57.490	99.10
	6	70K	0.308	44.61	69K	79.21	86.26	69K	**57.557**	106.79
	8	91K	0.249	57.89	90K	77.19	86.93	90K	54.385	131.57
	10	112K	**0.215**	73.49	111K	77.48	102.40	111K	57.221	173.74
	12	133K	0.225	87.08	132K	**80.18**	124.08	132K	57.270	206.73
SEA-2-HOP	4	182K	0.309	14.28	180K	76.30	43.51	48K	**86.768**	94.04
	6	265K	0.213	20.13	263K	77.27	59.82	69K	86.706	138.10
	8	347K	0.185	24.91	345K	76.61	79.56	90K	86.707	178.64
	10	430K	**0.159**	32.68	428K	78.38	95.69	111K	86.680	232.91
	12	513K	0.188	38.73	511K	**80.01**	112.93	132K	86.699	269.71
SEA-2-HOP-AUG	4	248K	0.444	16.86	248K	77.21	48.65	65K	84.889	124.96
	6	363K	0.350	24.84	363K	75.19	70.05	94K	85.141	203.38
	8	478K	0.285	31.48	476K	76.55	90.78	123K	86.660	270.85
	10	594K	0.205	39.25	594K	**79.08**	109.91	152K	**86.673**	363.58
	12	709K	**0.189**	46.51	707K	77.52	133.48	181K	86.614	421.46

Table 4: Influence of parameter k for the *SEA*-K-HOP model; best configuration for each model is highlighted in green

			ZINC		OGBG-MOLHIV		PATTERN	
Model	**#exp.**	***k***	**#prms**	**MAE**	**#prms**	**%ROC-AUC**	**#prms**	**%ACC**
SEA-K-HOP	6	2	265K	0.213	263K	**77.27**	69K	**86.768**
		3	266K	**0.191**	263K	76.15	69K	86.728
		4	266K	0.316	263K	73.48	69K	86.727
	10	2	430K	**0.159**	428K	**78.38**	111K	86.680
		3	433K	0.171	428K	74.67	111K	**86.765**
		4	433K	0.239	428K	73.72	111K	86.725

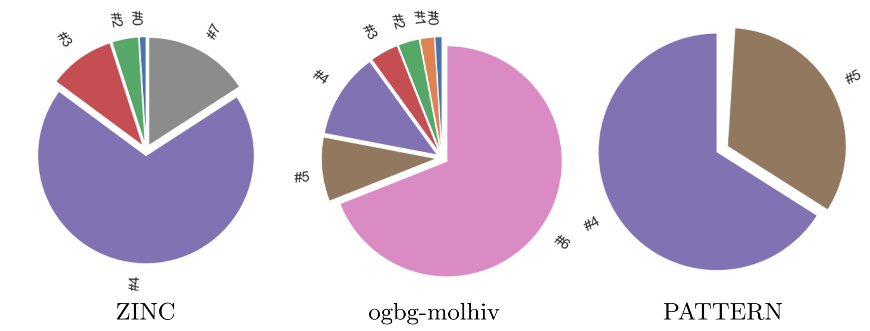
like in molecules (ZINC + ogbg-molhiv). In contrast to that is the behavior on graphs following the stochastic block model (PATTERN). On the latter one, the best performance could be observed by including k-hop information, whereas an aggregation yields too simplified features to be competitive. For the real-world molecules (ZINC + ogbg-molhiv) datasets, we observe that more experts boost the performance for the various SEA extensions.

5.4 Stretching Locality in SEA-K-HOP

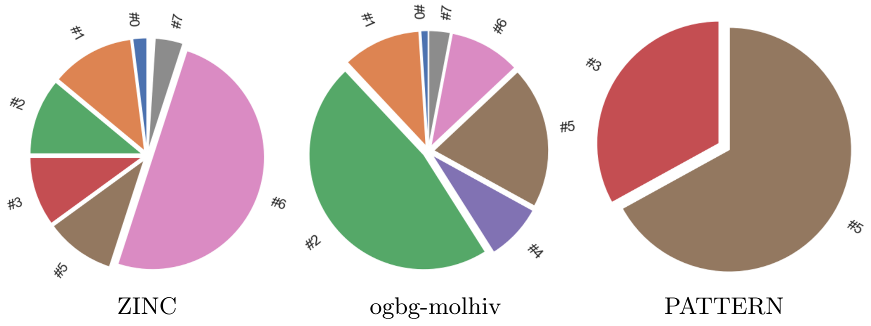
Lastly, we investigate the influence of the parameter k for the *SEA*-K-HOP model. Generally, by increasing the parameter k, the model diverges to the full model being also examined for the *SAN* architecture explained in [13]. In short, the full setting takes edges into account that is given by the input data and also sends information over non-existent edges, i.e., the argumentation is on a full graph setting. In our model, we smooth the transition from edges being given in the input data to the full setting that naturally arises when k, the number of hops, is set to a sufficiently high number. Table 4 summarizes the results for the non-augmented model, i.e., no extra linear layers are used for each k-hop neighborhood. The number of parameters stays the same by increasing k.

5.5 Distribution of Experts

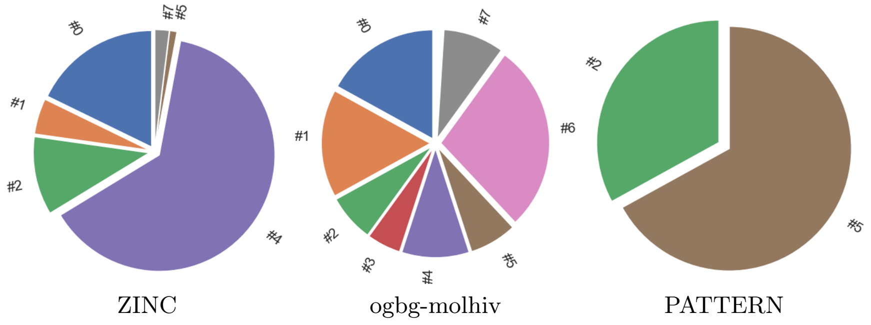
We evaluate the distributions of the experts being chosen to compute the nodes' representations in the following. We set the number of experts to 8. Figure 3 summarizes the relative frequencies of the experts being chosen on the datasets ZINC, ogbg-molhiv, and PATTERN. Generally, the performance of the 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 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 ϵ introduces randomness. Like a decaying



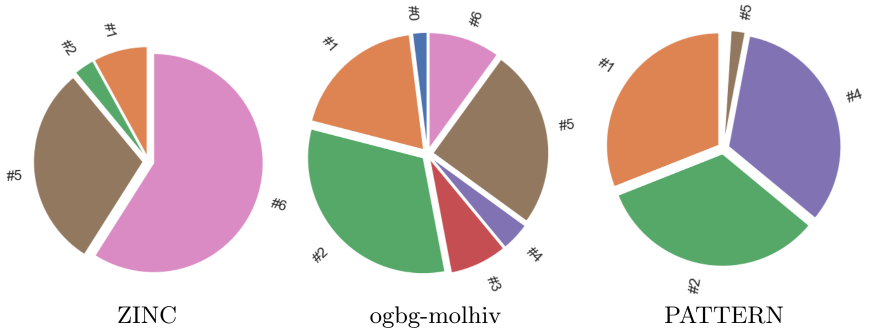
(a) SEA-GNN



(b) SEA-AGGREGATED



(c) SEA-2-HOP



(d) SEA-2-HOP-AUG

Fig. 3: 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.

greedy policy in Reinforcement Learning (RL), we choose a random expert with probability ϵ 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 experts in the molecular datasets - 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.

6 Conclusion

We introduced the theoretical foundation for integrating an expert heuristic within transformer-based graph neural networks. This opens a fruitful direction for future works that go beyond successive 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 steering 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* ***S****h****e****ll* ***A****ttention* (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 being more economically. Additionally, we report experiments that stress the number of graph shells that are taken into account.

Acknowledgements This work has been funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18050C (MLWin) and Grant No. 01IS18036A (MCML). The authors of this work take full responsibilities for its content.

References

1. Abbe, E.: Community detection and stochastic block models. Found. Trends Commun. Inf. Theory (Jun 2018). https://doi.org/10.1561/0100000067, https://doi.org/10.1561/0100000067

2. Beaini, D., Passaro, S., Létourneau, V., Hamilton, W.L., Corso, G., Liò, P.: Directional graph networks. CoRR (2020), https://arxiv.org/abs/2010.02863
3. Dwivedi, V.P., Bresson, X.: A generalization of transformer networks to graphs (2021)
4. Dwivedi, V.P., Joshi, C.K., Laurent, T., Bengio, Y., Bresson, X.: Benchmarking graph neural networks. arXiv preprint arXiv:2003.00982 (2020)
5. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. CoRR (2021), https://arxiv.org/abs/2101.03961
6. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems (2017), https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf
7. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bull. Amer. Math. Soc. (Aug 2006). https://doi.org/10.1090/s0273-0979-06-01126-8
8. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., Leskovec, J.: Open graph benchmark: Datasets for machine learning on graphs. arXiv preprint arXiv:2005.00687 (2020)
9. Hu, Z., Dong, Y., Wang, K., Sun, Y.: Heterogeneous graph transformer. In: Proceedings of The Web Conference 2020. WWW '20 (2020). https://doi.org/10.1145/3366423.3380027, https://doi.org/10.1145/3366423.3380027
10. Irwin, J.J., Sterling, T., Mysinger, M.M., Bolstad, E.S., Coleman, R.G.: Zinc: A free tool to discover chemistry for biology. Journal of chemical information and modeling (2012)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (ICLR) (2015)
12. Kipf, T.N., Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. In: Proceedings of the 5th International Conference on Learning Representations. ICLR '17 (2017), https://openreview.net/forum?id=SJU4ayYgl
13. Kreuzer, D., Beaini, D., Hamilton, W.L., Létourneau, V., Tossou, P.: Rethinking graph transformers with spectral attention (2021)
14. Nguyen, D.Q., Nguyen, T.D., Phung, D.: Universal self-attention network for graph classification. arXiv preprint arXiv:1909.11855 (2019)
15. Oono, K., Suzuki, T.: Graph neural networks exponentially lose expressive power for node classification. arXiv: Learning (2020)
16. Rong, Y., Bian, Y., Xu, T., Xie, W., WEI, Y., Huang, W., Huang, J.: Self-supervised graph transformer on large-scale molecular data. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) Advances in Neural Information Processing Systems (2020), https://proceedings.neurips.cc/paper/2020/file/94aef38441efa3380a3bed3faf1f9d5d-Paper.pdf
17. Schwartz, R., Dodge, J., Smith, N.A., Etzioni, O.: Green ai. Commun. ACM (Nov 2020). https://doi.org/10.1145/3381831, https://doi.org/10.1145/3381831
18. Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., Dean, J.: Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538 (2017)

19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, u., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS'17 (2017)
20. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. 6th International Conference on Learning Representations (2017)
21. 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., Zhang, Z.: Deep graph library: A graph-centric, highly-performant package for graph neural networks. arXiv preprint arXiv:1909.01315 (2019)
22. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems (2021). https://doi.org/10.1109/TNNLS.2020.2978386
23. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? CoRR (2018), http://arxiv.org/abs/1810.00826
24. Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Liu, T.Y.: Do transformers really perform bad for graph representation? arXiv preprint arXiv:2106.05234 (2021)
25. Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.J.: Graph transformer networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems (2019), https://proceedings.neurips.cc/paper/2019/file/9d63484abb477c97640154d40595a3bb-Paper.pdf
26. Zhou, D., Zheng, L., Han, J., He, J.: A Data-Driven Graph Generative Model for Temporal Interaction Networks (2020), https://doi.org/10.1145/3394486.3403082