Even Sparser Graph Transformers

Hamed Shirzad
University of British Columbia
shirzad@cs.ubc.ca

Honghao Lin
Carnegie Mellon University
honghaol@andrew.cmu.edu

Balaji Venkatachalam
Meta*
bave@meta.com

Ameya Velingker
Independent Researcher*
ameyav@gmail.com

David P. Woodruff
CMU & Google Research
dwoodruf@cs.cmu.edu

Danica J. Sutherland
UBC & Amii
dsuth@cs.ubc.ca

Abstract

Graph Transformers excel in long-range dependency modeling, but generally require quadratic memory complexity in the number of nodes in an input graph, and hence have trouble scaling to large graphs. Sparse attention variants such as Exphormer can help, but may require high-degree augmentations to the input graph for good performance, and do not attempt to sparsify an already-dense input graph. As the learned attention mechanisms tend to use few of these edges, such high-degree connections may be unnecessary. We show (empirically and with theoretical backing) that attention scores on graphs are usually quite consistent across network widths, and use this observation to propose a two-stage procedure, which we call Spexphormer: first, train a narrow network on the full augmented graph. Next, use only the active connections to train a wider network on a much sparser graph. We establish theoretical conditions when a narrow network's attention scores can match those of a wide network, and show that Spexphormer achieves good performance with drastically reduced memory requirements on various graph datasets. Code can be found at https://github.com/hamed1375/Sp_Exphormer.

1 Introduction

The predominant story of the last half-decade of machine learning has been the runaway success of Transformer models (Vaswani et al., 2017), across domains from natural language processing (Vaswani et al., 2017; Devlin et al., 2018; Zaheer et al., 2020) to computer vision (Dosovitskiy et al., 2020) and, more recently, geometric deep learning (Dwivedi and Bresson, 2020; Kreuzer et al., 2021; Ying et al., 2021; Rampášek et al., 2022; Shirzad et al., 2023; Müller et al., 2023). Conventional ("full") Transformers, however, have a time and memory complexity of $\mathcal{O}(nd^2 + n^2 d)$, where n is the number of entities (nodes, in the case of graphs), and d is the width of the network. Many attempts have been made to make Transformers more efficient (see Tay et al. (2020) for a survey on efficient transformers for *sequence modeling*). One major line of work involves *sparsifying* the attention mechanism, constraining attention from all $\mathcal{O}(n^2)$ pairs to some smaller set of connections. For instance, for sequential data, BigBird (Zaheer et al., 2020) constructs a sparse attention mechanism by combining sliding windows, Erdős-Rényi auxiliary graphs, and universal connectors. On the other hand, for graph data, Exphormer (Shirzad et al., 2023) constructs a sparse interaction graph consisting of edges from the input graph, an overlay expander graph, and universal connections. We refer to such a network as a *sparse attention network*.

Exphormer reduces each layer's complexity from $\mathcal{O}(nd^2 + n^2 d)$ to $\mathcal{O}((m+n)d^2)$, where n is the number of nodes, m is the number of interaction edges in the sparse attention mechanism, and d is

*Work done in part while at Google.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

https://doi.org/10.52202/079017-2277

the hidden dimension or width. Even so, training is still very memory-intensive for medium to large scale graphs. Also, for densely-connected input graphs with $\Theta(n^2)$ edges, there is no asymptotic improvement in complexity, as Exphormer uses all of the $\Theta(n^2)$ edges of the original input graph. Our goal is to scale efficient graph Transformers, such as Exphormer, to even larger graphs.

One general approach for scaling models to larger graphs is based on batching techniques. Prominent approaches include egocentric subgraphs and random node subsets (Wu et al., 2022, 2023, 2024). Egocentric subgraphs choose a node and include all of its k-hop neighbors; the expander graphs used in Exphormer, however, are exactly defined so that the size of these subgraphs grows exponentially in the number of layers – prohibitively expensive for larger graphs. A similar issue arises with universally-connected nodes, whose representation depends on all other nodes. For uniformly-random subset batching, as the number b of batches into which the graph is divided grows, each edge has chance $\frac{1}{b}$ to appear in a given step. Thus, b cannot be very large without dropping important edges. A similar problem can happen in random neighbor sampling methods such as GraphSAGE (Hamilton et al., 2017). Although this model works well on message-passing neural networks (MPNNs) which only use the graph edges, using it for expander-augmented graphs will select only a small ratio of the expander edges, thereby breaking the universality properties provided by the expander graph.

Expander graphs enable global information propagation, and when created with Hamiltonian cycles and self-loops, produce a model that can provably approximate a full Transformer (Shirzad et al., 2023, Theorem E.3). Yet not all of these edges turn out to be important in practice: we expect some neighboring nodes in the updated graph to have more of an effect on a given node than others. Thus, removing low-impact neighbors can improve the scalability of the model. The challenge is to identify low-impact edges without needing to train the (too-expensive) full model. Figure 1 illustrates advantages of this batching approach other; this is also discussed further in Appendix E.

One approach is to train a smaller network to identify which edges are significant. It is not obvious *a priori* that attention scores learned from the smaller network will estimate those in the larger network, but we present an experimental study verifying that attention scores are surprisingly consistent as the network size reduces. We also give theoretical indications that narrow networks are capable of expressing the same attention scores as wider networks of the same architecture.

Our approach. We first train a small-width network in order to estimate pairwise attention score patterns, which we then use to sparsify the graph and train a larger network. We first train the graphs without edge attributes. This reduces the complexity of Exphormer to $\mathcal{O}(md + nd^2)$ and then by training a much smaller width $d_s \ll d$ network, reduces the time and memory complexity by at least a factor of d/d_s. We also introduce two additions to the model to improve this consistency. Training this initial network can still be memory-intensive, but as the small width implies the matrix multiplications are small, it is practical to train this initial model on a CPU nodes with sufficient RAM (typically orders of magnitude larger than available GPU memory), without needing to use distributed computation. Once this initial model is trained, the attention scores can be used in creating a sparse graph, over which we train the second network. These initial attention scores can be used as edge features for the second network.

As mentioned previously, we use the attention scores obtained from the trained low-width network to sparsify the graph. By selecting a fixed number c of edges per attention layer for each node, we reduce the complexity of each layer to $\mathcal{O}(nd^2 + ndc)$. This sparsification alleviates the effect of a large number of edges, and allows for initial training with a larger degree expander graph, since most of the expander edges will be filtered for the final network. This sparsification differs from conventional graph sparsification algorithms (for MPNNs) in three ways. First, we use expander edges, self-loops, and graph edges and sparsify the combination of these patterns together. Second, this sparsification is layer-wise, which means that in a multi-layer network the attention pattern will vary from layer to layer. Finally, our sampling uses a smaller network trained on the same task, identifying important neighbors based on the task, instead of approaches independent of the task such as sampling based on PageRank or a neighbor's node degree.

Another advantage of this approach is that the fixed number of neighbors for each node enables regular matrix calculations instead of the edge-wise calculations used by Kreuzer et al. (2021); Shirzad et al. (2023), greatly improving the speed of the model. After this reduction, batching can be done based on the edges over different layers, enabling Transformers to be effectively batched while still effectively approximating the main Transformer model, enabling modeling long-range dependencies. In batching large graphs, sampling without replacement from attention edges with varying weights can be very

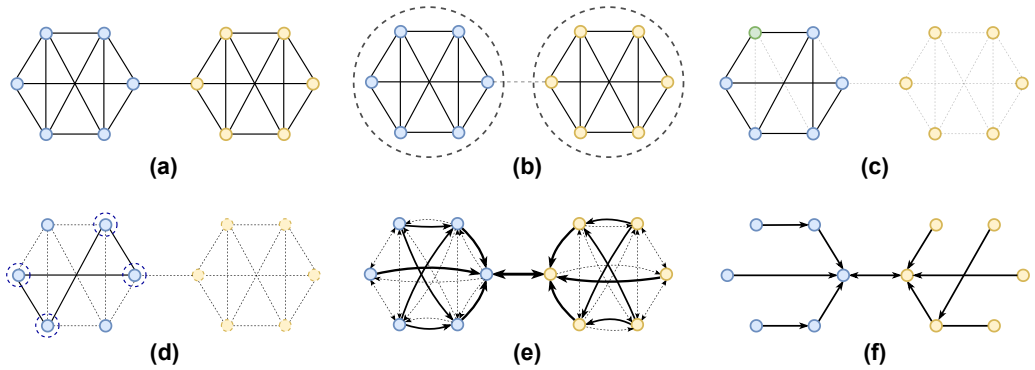

Figure 1: Figure (a) shows a very simple synthetic graph where each node has a binary classification task of determining whether there exists a node of the opposite color in the same connected component. This task requires learning long-range dependencies. Figure (b) shows a natural clustering of the graph. This clustering would mean no node can do its task if models are trained only on the clusters. Figure (c) shows a neighbor sampling starting from the green node, where random sampling fails to select the important edge that bridges the two different colored nodes. Figure (d) shows a random subset sampling strategy that the task is solvable if and if only the two sides of the bridge between the two colors get selected. Now if we keep increasing the size of both sides of the bridge, while keeping just one edge between two colors, the probability of selecting the bridge in any batch goes to zero, and thus the training will fail in this scenario. (e) shows attention scores between the nodes if trained with an attention-based network. Dashed lines have near zero attention scores, and thicker lines indicate a larger attention score. Knowing these attention scores will mean each node with just one directional edge can do the task perfectly. The attention edges are shown in (f). In case two nodes are equally informative selecting either of them leads to the correct result.

slow. This is especially true if the attention scores are highly concentrated on a small number of neighbors for most of the nodes. Sequential sampling of neighbors can be very slow, while parallel sampling can lead to many conflicts in this scenario. We use reservoir sampling, enabling parallel sampling with an easy, efficient GPU implementation, improving the sampling process significantly.

We only use the Transformer part of the Exphormer model, not the dual MPNN+Transformer architecture used by Shirzad et al. (2023); Rampášek et al. (2022). Unlike the Exphormer approach, we do not assume that the expander graph is of degree $\mathcal{O}(1)$; we can see this as interpolating between MPNNs and full Transformers, where smaller degree expander graphs mostly rely on the graph edges and are more similar to MPNNs, while higher degree expander graphs can resemble full attention, in the most extreme case of degree $n-1$ exactly recovering a full Transformer.

To summarize, the contributions of this paper are as follows: 1.) We experimentally and theoretically analyze the similarity of attention scores for networks of different widths, and propose two small architectural changes to improve this similarity. 2.) We propose layer-wise sparsification, by sampling according to the learned attention scores, and do theoretical analysis on the sparsification guarantees of the attention pattern. 3.) Our two-phase training process allows us to scale Transformers to larger datasets as it has significantly smaller memory consumption, while maintaining competitive accuracy.

2 Related Work

Graph Transformer Architectures. Attention mechanisms were proposed in early (message-passing) Graph Neural Network (GNN) architectures such as Graph Attention Networks (GAT) (Veličković et al., 2018), where they guide node aggregation among neighbors, without using positional encodings. GraphBert (Zhang et al., 2020) finds node encodings based on the underlying graph structure. Subsequent work has proposed full-fledged graph Transformer models that generalize sequence Transformers (Dwivedi and Bresson, 2020) and are not limited to message passing between nodes of the input graph; these include Spectral Attention Networks (SAN) (Kreuzer et al., 2021), Graphormer (Ying et al., 2021), GraphiT (Mialon et al., 2021), etc. GraphGPS (Rampášek et al., 2022) combines attention mechanisms with message passing, allowing the best of both worlds.

https://doi.org/10.52202/079017-2277

Efficient Graph Transformers. Several recent works have proposed various scalable graph transformer architectures. NAGphormer (Chen et al., 2022a) and Gophormer (Zhao et al., 2021) use a sampling-based approach. On the other hand, Difformer (Wu et al., 2023) proposes a continuous time diffusion-based transformer model. Exphormer (Shirzad et al., 2023) proposes a sparse graph that combines the input graph with edges of an expander graph as well as virtual nodes. They show that their model works better than applying other sparse Transformer methods developed for sequences. Another work, NodeFormer (Wu et al., 2022), which is inspired by Performer (Choromanski et al., 2021), uses the Gumbel-Softmax operator as a kernel to efficiently propagate information among all pairs of nodes. SGFormer (Wu et al., 2024) shows that just using a one layer transformer network can sometimes improve the results of GCN-based networks and the low memory footprint can help scale to large networks. Perhaps most conceptually similar to our work is Skeinformer (Chen et al., 2022b), which uses sketching techniques to accelerate self-attention.

Sampling and batching techniques. Some sampling-based methods have been used to alleviate the problem of "neighborhood explosion." For instance, sampling was used in GraphSAGE (Hamilton et al., 2017), which used a fixed-size sample from a neighborhood in the node aggregation step. GraphSAINT (Zeng et al., 2020) scales GCNs to large graphs by sampling the training graph to create minibatches.

Other. Expander graphs were used in convolutional networks by Prabhu et al. (2018).

3 Preliminaries and Notation

Exphormer. EXPHORMER is an expander-based sparse attention mechanism for graph transformers that uses $O(|V| + |E|)$ computation, where $G = (V, E)$ is the underlying input graph. Exphormer creates an interaction graph H that consists of three main components: edges from the input graph, an overlaid expander graph, and virtual nodes (which are connected to all the original nodes).

For the *expander graph* component, Exphormer uses a constant-degree random expander graph, with $\mathcal{O}(n)$ edges. Expander graphs have several useful theoretical properties related to spectral approximation and random walk mixing, which allow the propagation of information between pairs of nodes that are distant in the input graph G without explicitly connecting all pairs of nodes. The expander edges introduce many alternative short paths between the nodes and avoid the information bottleneck that can be caused by the virtual nodes.

Our model. We use H to denote the attention pattern, and $\mathcal{N}_H(i)$ the neighbors of node i under that pattern. Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^{d \times n}$ be the matrix of d-dimensional embeddings for all of the n nodes. Our primary "driver" is then h-head attention: using $\odot$ for element-wise multiplication,

$$\text{ATTN}_H(\mathbf{X})_{:,i} = \mathbf{x}_i + \sum_{j=1}^{h} \mathbf{V}_i^j \cdot \sigma\left(\left(\mathbf{E}^j \odot \mathbf{K}^j\right)^T \mathbf{Q}_i^j + \mathbf{B}^j\right),$$

where $\mathbf{V}_i^j = \mathbf{W}_V^j \mathbf{X}_{\mathcal{N}_H(i)}$, $\mathbf{K} = \mathbf{W}_K^j \mathbf{X}_{\mathcal{N}_H(i)}$, and $\mathbf{Q}_i^j = \mathbf{W}_Q^j \mathbf{x}_i$, are linear mappings of the node features for the neighbors $\mathbf{X}_{\mathcal{N}_H(i)}$, and $\mathbf{E}^j = \mathbf{W}_E^j \mathcal{E}_{\mathcal{N}_H(i)}$ and $\mathbf{B}^j = \mathbf{W}_B^j \mathcal{E}_{\mathcal{N}_H(i)}$ are linear maps of the edge features $\mathcal{E}$, which is a $d_E \times |\mathcal{N}_H(i)|$ matrix of features for the edges coming in to node i. Exphormer uses learnable edge features for each type of added edge, and original edge features for the graph's edges. If the graph does not have any original edge features, it uses a learnable edge feature across all graph edges. Edge features help the model distinguish the type of attention edges. Here, σ is an activation function. In both Exphormer and our work the activation function is ReLU.

In the absence of edge features, which is the case for most of the transductive datasets, including the datasets that have been used in this paper, $\mathcal{E}_e$ for any attention edge e can have one of three possible representations, and so $\mathbf{E}^j$ can be computed more simply by first mapping these three types of edge features with $\mathbf{W}_E^j$ for head j, and then replacing the mapped values for each edge type. This simple change reduces the complexity of the Exphormer from $\mathcal{O}(md^2 + nd^2)$ to $\mathcal{O}(md + nd^2)$.

Compared to prior work, we introduce $\mathbf{B}^j$ as a simpler route for the model to adjust the importance of different edge types. Considering Exphormer as an interpolation between MPNNs and full Transformers, the $\mathbf{B}^j$ model has an easier path to allow for attention scores to be close to zero for all non-graph attention edges, without restricting the performance of the attention mechanism on

graph edges. Consequently, it can function roughly as an MPNN (similar to GAT) by zeroing out the non-local attention paths. We use $d_E = d$, and have each layer output features of the same width as its input, so that each of the $\mathbf{W}_\cdot^j$ parameter matrices except for $\mathbf{W}_B^j$ are $d \times d$, and $\mathbf{W}_B^j$ is $d \times 1$.

As a simple illustration that $\mathbf{E}^j$ is insufficient to allow near-zero attention scores, thus highlighting the importance of $\mathbf{B}^j$, note that if the columns of $\mathbf{K}$ and $\mathbf{Q}$ are distributed independently and uniformly on a unit ball (e.g., under a random initialization), there is no vector $\mathbf{E}^j$ which is identical for all edges of an expander graph that can make the attention scores for all the expander edges near-zero.

Our network compared to Exphormer. We use Exphormer as the base model because it provides us the flexibility to adjust the sparsity of the attention graph and to interpolate between MPNNs and full Transformers. Exphormer can model many long-range dependencies that are not modeled by MPNNs and are very expensive to model in a full Transformer. For example, one cannot train a full Transformer model in the memory of a conventional GPU device for a dataset such as Physics, which has a graph on just 34K nodes. In our instantiation of Exphormer, we add self-loops for every node and use $d/2$ random Hamiltonian cycles to construct our expander graph as described in (Shirzad et al., 2023, Appendix C.2). We do not add virtual nodes in our networks (note that the resulting network is still a universal approximator; Shirzad et al., 2023, Theorem E.3). Although the best known results for Exphormer combine sparse attention with MPNNs, in this work, we avoid the MPNN component for scalability reasons. We also make two additional changes, see Section 4.

4 Method

Our method consists of a two-phase training process. The first phase trains a model we call the *Attention Score Estimator Network*, whose goal is to estimate the attention scores for a larger network. This model is not particularly accurate; its only goal is for each node to learn which neighbors are most important. The learned attention scores for each layer of the first network are then used to construct sparse interaction graphs for each layer in a second model, which is trained (with hyperparameter tuning for the best results) and serves as the final predictor.

Attention Score Estimator Network. For this network, we use a width of 4 or 8, with just one attention head, in our training. We tune the other hyperparameters in order to have a converged training process with reasonably high accuracy, but we do not spend much time optimizing this network as it is sufficient to learn the important neighbors for each node, i.e., edges with high attention scores. This network will be trained with as many layers as the final network we want to train. Because it is so narrow, it has many fewer parameters and hence much less memory and time complexity, making it cheaper to train. Moreover, we only need to do this training once per number of layers we consider, conditioned on the fact that the training converges, even if the final model has a large number of hyperparameters. Compared to Exphormer, we use a much higher-degree expander graph: 30 to 200 instead of the 6 used for most transductive graphs by Shirzad et al. (2023). As most of the considered datasets do not have edge features, we use a learnable embedding for each type of edge (graph edge, expander edge, or self-loop). We also make two small changes to the architecture and the training process of this model, discussed below. Section 5 shows experimentally that the low-width network is a good estimator of the attention scores for a large-width network.

Normalizing V. Having a smaller attention score, $\alpha_{ij} < \alpha_{ij'}$, does not necessarily mean that j's contribution to i's new features is smaller than that of j': if $\|\mathbf{V}_j\| \gg \|\mathbf{V}_{j'}\|$, the net contribution of j could be larger. Although Transformers typically use layer normalization, they do not typically do so after mapping X to $\mathbf{V}$. We normalize the rows of $\mathbf{V}$ to have the same vector sizes for all nodes. In our experiments, normalizing to size one reduced performance significantly; however, adding a learnable global scale s, so that $\mathbf{V}_i$ becomes $\frac{s\mathbf{V}_i}{||\mathbf{V_i}||_2}$, maintained performance while making attention scores more meaningful.

Variable Temperature One of the side goals is to have sharper attention scores, guiding the nodes to get their information from as few nodes as possible. Using temperature in the attention mechanism can do this, where logits will be divided by a temperature factor τ before being fed into a softmax. Normal attention corresponds to $\tau = 1$; smaller τ means sharper attention scores. However, setting the temperature to a small value from the beginning will make the random initialization more significant, and increase the randomness in the training process. Instead, we start with $\tau = 1.0$ and gradually anneal it to 0.05 by the end of the training. We set an initial phase for λ epochs

 https://doi.org/10.52202/079017-2277

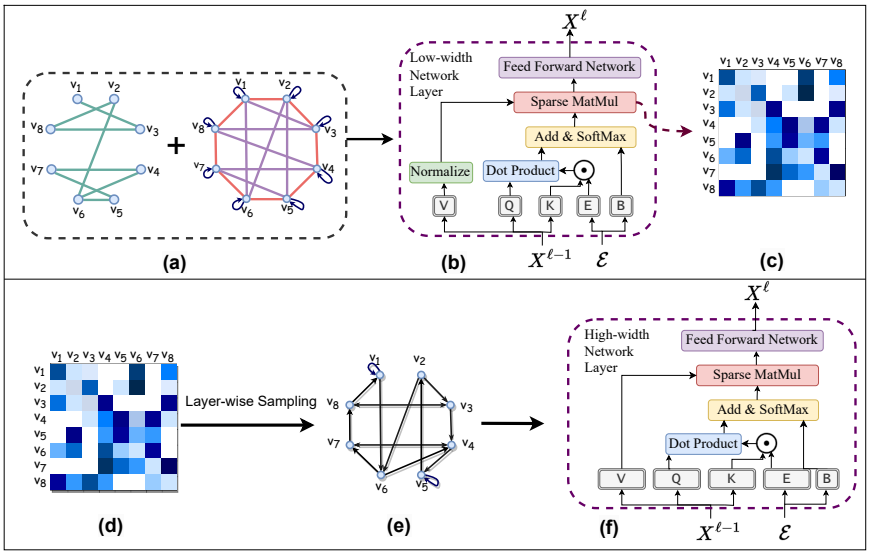

Figure 2: Steps of our method. **(a)** The attention mechanism for the attention score estimator network combines graph edges with an expander graph and self-loops. The expander graphs are constructed by combining a small number of Hamiltonian cycles – here two, in red and in purple – then confirming the spectral gap is large enough. **(b)** Self-attention layers in the estimator network use this sparse attention mechanism; its self-attention layers normalize $\mathbf{V}$. **(c, d)** Attention scores are extracted from this network for each layer, and used to sample, in **(e)**, a sparse directed graph, which becomes the attention graph for the final network **(f)**. This network, with a much larger feature dimension, does not normalize $\mathbf{V}$.

where we use $\tau = 1$; this lets the model learn which neighbors are more important for each node slowly. We multiply τ with a factor γ after each epoch, obtaining a temperature in epoch $t > \lambda$ of $\max(\gamma^{t-\lambda}, 0.05)$. We use $\lambda = 5$ and $\gamma = 0.99$ or 0.95 depending on how fast the learning converges.

Sparser Attention Pattern. The memory and time complexity of Exphormer is linearly dependent on the number of edges. Also, with a small number of layers, the expander degree should be high enough to ensure a large enough receptive field for each node in order to learn the long-range dependencies. Not all these edges are equally important, and many of them will have a near-zero effect on the final embedding of each node. Reducing the number of edges can alleviate memory consumption. Additionally, a sparser pattern lets us use batching techniques for the larger graphs. In this work, we analyze how effectively the sparser model can work and up to what factor we can sparsify. For each layer, e.g., ℓ, we select a $\deg_\ell$ as a fixed degree for each node and sample without replacement according to the attention score estimator network's attention scores in each epoch of training or evaluation. Having the same degree for each node's attention pattern also means that attention can be calculated using (much-more-optimized) standard matrix multiplications, rather than the propagation techniques used in Exphormer and SAN (Kreuzer et al., 2021).

To sparsify the graph, in each epoch, we sample a new set of edges according to the learned attention scores from the smaller network. The reason why we do this rather than a simpler strategy such as selecting top-scored edges is that in many cases, several nodes can have very similar node features. If we assume nodes $u_1, u_2, \ldots, u_p$ from the neighbors of node v have almost the same features, and if the attention scores for these nodes are $\alpha_1, \alpha_2, \ldots, \alpha_p$, any linear combination of $\sum_{i=1}^{p} \alpha_i = \alpha$ will lead to the same representation for node v. If features are exactly the same, α will be divided between these nodes, and even if α is large, each node's attention score from v can be small. By sampling, we have a total α chance of selecting any of the nodes $u_{1:p}$. In each epoch, we re-sample a new set of edges for each node from its original neighborhood.

Faster Sampling Using Reservoir Sampling. Sampling without replacement using default library calls is very slow, especially if few neighbors dominate the attention scores. We instead use reservoir sampling (Efraimidis and Spirakis, 2006), which is GPU-friendly and parallelizable. For reservoir sampling of k neighbors from the neighborhood of node i, with attention scores $\mathbf{a} = (a_1, a_2, \cdots, a_{|\mathcal{N}_H(i)|})$, we first do a uniform random sampling of $\mathbf{u} = (u_1, u_2, \cdots, u_{|\mathcal{N}_H(i)|})$, where the u_i are i.i.d. samples from $\mathrm{Uniform}(0, 1)$. Then we calculate $\frac{1}{\mathbf{a}} \log(\mathbf{u})$ with element-wise

multiplication, and select the indices with the top k values from this list. Selecting k-th rank from n values and pivoting has a worst-case $\mathcal{O}(n)$ time algorithm, which is much faster than the $\mathcal{O}(nk)$ worst case time for trial-and-error. Pseudocode is given in Algorithm 1. The GPU-friendly version of this can be implemented by sampling for nodes in parallel, but requires forming a regular matrix for the attention scores, which can be done by extending each attention score vector by the maximum degree, or selecting a value $k' \gg k$ and first sampling k' and selecting the top k' attention scores from each node, making sure that the sum of the rest of the neighbor's attention scores are very near to zero. Then by forming a rectangular attention matrix, uniform sampling and element-wise multiplications are much faster on GPU, and sampling from the entire batch is much more efficient.

Algorithm 1 Reservoir Sampling from a Node's Neighborhood

Input: Attention scores $\mathbf{a} = a^{(\ell)}_{i,\mathcal{N}_H(i)}$, number of neighbors to sample: $\deg_\ell$
Output: List of $\deg_\ell$ neighbors of node i

function RESERVOIRSAMPLE($\mathbf{a}, \deg_\ell$)
    $\mathbf{u} \sim \text{Uniform}(0,1)^{|\mathcal{N}_{H(i)}|}$
    return $\text{argtop}_{\deg_\ell}(\frac{1}{\mathbf{a}}\log(\mathbf{u}))$
end function

Batching. Each batch starts with a random subset of "target" nodes B. These are the nodes whose last-layer representations we will update in this optimization step. To calculate these representations, we need keys and values based on the previous layer's representations for the relevant neighbors of each target node (again, sampling neighbors from the graph augmented by an expander graph). To approximate this, we sample $\deg_L$ neighbors for each target node. Then we have a set of at most $|B|(\deg_L + 1)$ nodes whose representations we need to calculate in layer $L-1$; we repeat this process, so that in layer ℓ we need to compute representations for up to $\mathcal{Q}^{(\ell)} \leq \min(|B| \prod_{i=\ell+1}^{L}(\deg_i + 1), n)$ query nodes, with $|\mathcal{Q}^{(\ell)}| \deg_\ell$ attention edges. Pseudocode is given in Algorithm 2.

When the number of layers L and degree $\deg_\ell$ are not too large, this batching can be substantially more efficient than processing the entire graph. Moreover, compared to other batching techniques, our approach selects neighbors *according to their task importance*. Except for optimization dynamics in the training process corresponding to minibatch versus full-batch training, training with batches is identical to training with the entire sparsified graph; if we choose a large $\deg_\ell$ equal to the maximum degree of the augmented graph, this is exactly equivalent to SGD on the full graph, without introducing any biases in the training procedure. This is in stark contrast to previous approaches, as illustrated in Figure 1. Unlike these prior approaches, which typically use the full graph at inference time, we can run inference with batch size as small as one (trading off memory for computation).

Algorithm 2 Neighborhood Sampling for a Batch of Nodes

Input: Attention scores in each layer: $\mathbf{a} = \left\{a^{(\ell)}_{i,j} \mid \forall i \in V, j \in \mathcal{N}_H(i), , 1 \leq \ell \leq L\right\}$, number of neighbors to sample in each layer: $\mathbf{deg} = \{\deg_1, \cdots, \deg_L\}$, and a batch of nodes $B \subseteq V$
Output: $\mathcal{Q}^{(\ell)}, \mathcal{K}^{(\ell)}, \mathcal{V}^{(\ell)}$, query, key, and value nodes in each layer

function SAMPLENEIGHBORHOOD($B, \mathbf{a}, \mathbf{deg}$)
    $\mathcal{V}^{(L+1)} \leftarrow B$
    for $\ell \leftarrow L$ to 1 **do**
        $\mathcal{Q}^{(\ell)} \leftarrow \mathcal{V}^{(\ell+1)}$
        for $i \leftarrow i \in \mathcal{Q}^{(\ell)}$ **do**
            $\mathcal{K}^{(\ell)}_i \leftarrow$ RESERVOIRSAMPLE($\mathbf{a}_{i,\mathcal{N}_H(i)}, \deg_\ell$)
        end for
        $\mathcal{K}^{(\ell)} \leftarrow \bigcup_{i \in \mathcal{Q}^\ell} \mathcal{K}^{(\ell)}_i$
        $\mathcal{V}^{(\ell)} \leftarrow \mathcal{Q}^{(\ell)} \bigcup \mathcal{K}^{(\ell)}$
    end for
    return $\left\{\left(\mathcal{V}^{(\ell)}, \mathcal{Q}^{(\ell)}, \mathcal{K}^{(\ell)}\right) \mid 1 \leq \ell \leq L\right\}$
end function

https://doi.org/10.52202/079017-2277

Fixed Node Degree Layers. Sparse matrix operations are not yet nearly as efficient as dense operations on GPU devices. Exphormer and SAN use a `gather` operation, which is memory-efficient but not time-efficient on a GPU (Zaheer et al., 2020). By normalizing the degree, instead of having $|\mathcal{Q}^{(\ell)}| \deg_\ell$ separate dot products between the query and key vectors, we can reshape the key vectors to be of size $|\mathcal{Q}^{(\ell)}| \times \deg_\ell \times d$ and the query is of shape $|\mathcal{Q}^{(\ell)}| \times d$. Now the dot product of query and key mappings can be done using $|\mathcal{Q}^{(\ell)}|, \deg_\ell \times d$ by $d \times 1$ matrix multiplications. This same size matrix multiplication can be done using highly optimized batch matrix multiplication operations in e.g. PyTorch and Tensorflow (Paszke et al., 2019; Abadi et al., 2015).

Theoretical Underpinnings. We first study the approximability of a network with a smaller hidden dimension or width. Formally, suppose that the width of a wide network is D. Then there exists a network with narrow dimensions for $\mathbf{W}_Q$ and $\mathbf{W}_K$, of dimension $\mathcal{O}(\frac{\log n}{\varepsilon^2}) \times D$ instead of $D \times D$, whose attention scores agree with those of the wide network up to $\mathcal{O}(\varepsilon)$ error (Theorem D.4). This reduction helps with the most intensive part of the calculation; others are linear with respect to the number of nodes n. While this is not the model we use in practice, Shirzad et al. (2024, Section 4) explore some scenarios common in graph Transformers that allow for the existence of "fully" narrow networks with accurate attention scores. They support these claims with experiments that show compressibility for some datasets we use. This is an existence claim; we will justify experimentally that in practice, training a narrow network does approximate attention scores well.

We then study the sampling procedure of our sparsification method. Under certain assumptions, we show that sampling roughly $\mathcal{O}(n \log n/\varepsilon^2)$ entries of the attention matrix A (corresponding to sampling this many edges in the graph) suffices to form a matrix B with $\|A - B\|_2 \leq \varepsilon \|A\|_2$, if we can access the entries of A (Theorem D.5). We cannot actually access the matrix A, but we do have attention scores A' from a narrow network. We show that if the entries of A are not seriously under-estimated by A', the same bound on the number of samples still holds (Proposition D.7).

5 Experimental Results

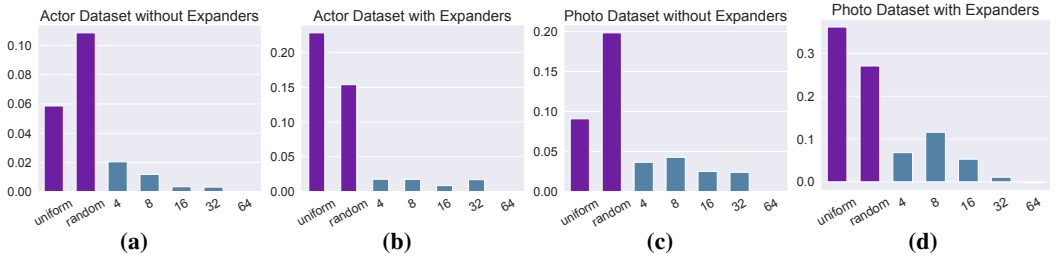

Figure 3: Energy distance between the attention scores of various networks to a network of width 64. "Uniform" refers to the baseline placing uniform scores on each neighbor, while "random" refers to the baseline with uniformly distributed logits. The remaining bars refer to networks trained on the appropriately labeled width.

Attention Score Estimation. To show how well the smaller network estimates the attention scores for a larger network, we conduct experiments on two smaller datasets, where we can reasonably train the full network at higher width for many runs to be able to have an estimation on the distribution of the attention scores. To this end, we use Actor (Lim et al., 2021) and Photo (Shchur et al., 2018) datasets. We train the network for hidden dimensions, h varying from 4 to 64 for both datasets. For each h we train the network 100 times. We consider the distribution of attention scores for each node and estimate the energy distance (Székely and Rizzo, 2013; an instance of the maximum mean discrepancy, Sejdinovic et al., 2013) for that node across each pair of h sizes.

We ran this experiment both when learning with just graph edges, and when adding expander and self-loop edges. It might be that the model, just by examining the category of the edges, may give a lower score to one type, making distributions seem more similar despite not identifying a small number of important neighbors as we want. However, in the presence of only one type of edge, the model can still consistently estimate which nodes should have a higher attention score.

We compare attention scores from our model with the uniform distribution on the neighbors (each neighbor of node i has score $\frac{1}{d_i}$), and to a distribution with logits uniform over $[-8, 8]$. The choice of

8 here is because in the network we clip the logits with an absolute value higher than 8. Figure 3 shows that even width-4 networks provide far superior estimates of attention scores than these baselines.

Table 1: Comparison of our model with other GNNs on five homophilic and three heterophilic datasets. The reported metric is ROC-AUC ($\times 100$) for the Minesweeper and Tolokers datasets and accuracy for all others. The average edge ratio is the degree of the nodes in Spexphormer over the average degree of the nodes in the initial attention pattern.

Model	Computer	Photo	CS	Physics	ogbn-arxiv
GCN	89.65 ± 0.52	92.70 ± 0.20	92.92 ± 0.12	96.18 ± 0.07	71.74 ± 0.29
GraphSAGE	91.20 ± 0.29	94.59 ± 0.14	93.91 ± 0.13	96.49 ± 0.06	71.49 ± 0.27
GAT	90.78 ± 0.13	93.87 ± 0.11	93.61 ± 0.14	96.17 ± 0.08	72.01 ± 0.20
GraphSAINT	90.22 ± 0.15	91.72 ± 0.13	94.41 ± 0.09	96.43 ± 0.05	68.50 ± 0.23
NodeFormer	86.98 ± 0.62	93.46 ± 0.35	95.64 ± 0.22	96.45 ± 0.28	59.90 ± 0.42
GraphGPS	91.19 ± 0.54	95.06 ± 0.13	93.93 ± 0.12	97.12 ± 0.19	70.92 ± 0.04
GOAT	90.96 ± 0.90	92.96 ± 1.48	94.21 ± 0.38	96.24 ± 0.24	72.41 ± 40
Exphormer+GCN	91.59 ± 0.31	95.27 ± 0.42	95.77 ± 0.15	97.16 ± 0.13	72.44 ± 0.28
Exphormer	91.16 ± 0.26	95.36 ± 0.17	95.19 ± 0.26	96.40 ± 0.20	71.27 ± 0.27
Spexphormer	91.09 ± 0.08	95.33 ± 0.49	95.00 ± 0.15	96.70 ± 0.05	70.82 ± 0.24
Avg. Edge Percent	7.6%	8.2%	12.8%	11.3%	13.7%

Model	Actor	Minesweeper	Tolokers
GloGNN	36.4 ± 1.6	51.08 ± 1.23	73.39 ± 1.17
GCN	33.23 ± 1.16	89.75 ± 0.52	83.64 ± 0.67
GraphGPS	37.1 ± 1.5	90.63 ± 0.67	83.71 ± 0.48
NAGphormer	-	84.19 ± 0.66	78.32 ± 0.95
NodeFormer	36.9 ± 1.0	86.71 ± 0.88	78.10 ± 1.03
GOAT	-	81.09 ± 1.02	83.11 ± 1.04
Difformer	36.5 ± 0.7	90.89 ± 0.58	83.57 ± 0.68
Exphormer+GAT	38.68 ± 0.38	90.74 ± 0.53	83.77 ± 0.78
Exphormer	39.01 ± 0.69	92.26 ± 0.56	83.53 ± 0.28
Spexphormer	38.59 ± 0.81	90.71 ± 0.17	83.34 ± 0.31
Avg. Edge Percent	5.8%	17.8%	8.9%

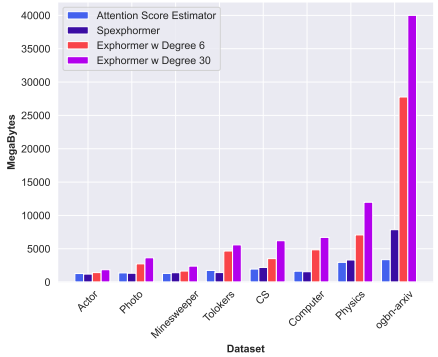

Figure 4: Memory usage comparison: Attention Score Estimator network and Spexphormer vs. Exphormer with expander degrees 6 and 30. Exphormer with degree 30 for the ogbn-arxiv dataset could not fit into the memory of a 40GB GPU device, and thus the number is a lower bound here.

Model	ogbn-proteins	Amazon2M	Pokec
MLP	72.04 ± 0.48	63.46 ± 0.10	60.15 ± 0.03
GCN	72.51 ± 0.35	83.90 ± 0.10	62.31 ± 1.13
SGC	70.31 ± 0.23	81.21 ± 0.12	52.03 ± 0.84
GCN-NSampler	73.51 ± 1.31	83.84 ± 0.42	63.75 ± 0.77
GAT-NSampler	74.63 ± 1.24	85.17 ± 0.32	62.32 ± 0.65
SIGN	71.24 ± 0.46	80.98 ± 0.31	68.01 ± 0.25
NodeFormer	77.45 ± 1.15	87.85 ± 0.24	70.32 ± 0.45
SGFormer	79.53 ± 0.38	89.09 ± 0.10	73.76 ± 0.24
Spexphormer	$\mathbf{80.65 \pm 0.07}$	$\mathbf{90.40 \pm 0.03}$	$\mathbf{74.73 \pm 0.04}$
Memory Information for Spexphormer			
Memory (MB)	2232	3262	2128
Batch Size	256	1000	500
Hidden Dimension	64	128	64
Number of layers	2	2	2
Number of Parameters	79,224	300,209	83,781

Table 2: Comparative results on large graph datasets, with ROC-AUC($\times 100$) reported for the ogbn-proteins dataset and accuracy for all others. GPU memory usage, batch sizes, hidden dimensions used to obtain these numbers, and the total number of parameters have been added at the bottom of the table.

Model Quality. We conduct experiments on eight medium-sized graphs, including five homophilic datasets: CS, Physics, Photo, Computer (Shchur et al., 2018) and ogbn-arxiv (Hu et al., 2021); and three heterophilic datasets: Minesweeper, Tolokers (Platonov et al., 2023), and Actor (Lim et al., 2021). For the CS, Physics, Photo, and Computer datasets, we use a random train/validation/test split of 60%/20%/20%. For ogbn-arxiv we follow the standard data split provided by the original source. For the Actor dataset, we use a 50%/25%/25% split following Wu et al. (2022). For the Minesweeper and Tolokers datasets, we use the standard split from Platonov et al. (2023). Results for these experiments are provided in Table 1.

In these medium-sized datasets, we are able to train the full Exphormer model. Our goal is to determine the extent of performance reduction when using two memory-efficient networks to estimate the original network. Results show that the two memory-efficient networks can efficiently estimate the original network, enabling us to scale the Exphormer to larger graph datasets. We compare the maximum required memory of the attention score estimator and final networks with that of the corresponding Exphormer model in Figure 4.

We then experiment on large graph datasets (ogbn-proteins, Amazon2M (Hu et al., 2021) and Pokec (Takac and Zabovsky, 2012)). The results provided in Table 2, demonstrate superior performance of our model despite limited memory constraints. We follow the standard data split for the ogbn-proteins dataset and follow Wu et al. (2024) for the dataset split on the Amazon2M and Pokec datasets, with 10%/10%/80% and 50%/25%/25% train/validation/test ratios. We emphasize that this split differs from the original dataset split used by many other works, making those numbers incomparable.

In all our experiments, we train the smaller network once, and then for the second network, we always use the same initial network's learned attention scores. Attention scores are collected from the network training step with the highest validation accuracy.

https://doi.org/10.52202/079017-2277

Table 3: Ablation studies on two homophilic and two heterophilic datasets. Metrics: accuracy for Photo and Computer, ROC-AUC (×100) for Tolokers and Minesweeper. For the initial network, we report the result for the network used for training the Spexphormer and thus, there is no confidence interval for them.

Model/Dataset	Computer	Photo	Minesweeper	Tolokers
Initial Network	85.23	91.70	85.67	80.16
Spexphormer-uniform	86.65 ± 0.46	94.21 ± 0.22	84.15 ± 0.22	82.56 ± 0.17
Spexphormer-max	89.31 ± 0.31	95.07 ± 0.20	87.92 ± 0.26	80.85 ± 0.23
Spexphormer w.o. temp	89.05 ± 0.35	95.30 ± 0.16	90.02 ± 0.02	83.34 ± 0.13
Spexphormer w.o. layer norm	89.70 ± 0.25	94.91 ± 0.18	89.65 ± 0.10	$\mathbf{84.06 \pm 0.10}$
Spexphormer	$\mathbf{91.09 \pm 0.08}$	$\mathbf{95.33 \pm 0.49}$	$\mathbf{90.71 \pm 0.17}$	83.34 ± 0.13

We use a subset of the following models in each of our tables as baselines, depending on the type of the dataset and scalability level of the models, GCN (Kipf and Welling, 2016), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), GraphSAINT (Zeng et al., 2020), Nodeformer (Wu et al., 2022), Difformer (Wu et al., 2023), SGFormer (Wu et al., 2024), GraphGPS (Rampášek et al., 2022), GOAT (Kong et al., 2023), GloGNN (Li et al., 2022), SGC (Wu et al., 2019), NAGphormer (Chen et al., 2022a), Exphormer (Shirzad et al., 2023), SIGN (Frasca et al., 2020). We borrow most of the baseline numbers in the tables from Wu et al. (2024); Deng et al. (2024).

Ablation Studies. We benchmark the effect of different parts of the model in Table 3. Spexphormer-uniform, rather than sampling based on the estimated attention scores, samples uniformly from the augmented graph; this is always worse than sampling, but the gap is larger for some datasets than others. Spexphormer-max takes the edges with the highest attention scores, rather than sampling; this again performs somewhat worse across datasets. Spexphormer w.o. temp uses a constant temperature of 1 in the initial attention score estimator network; Spexphormer w.o. layer norm removes our added layer normalization. these changes are smaller, and in one case layer normalization makes the results worse. Across the four datasets, however, it seems that both temperature and layer norm help yield more informative and sparser attention scores.

6 Conclusion & Limitations

We analyzed the alignment of the attention scores among models trained with different widths. We found that usually the smaller network's attention score distributions align with the larger network's. We also theoretically analyzed the compressibility of the larger Graph Transformer models. Based on these observations, we used a sampling algorithm to sparsify the graph on each layer and sampled a smaller number of edges per layer. As a result of these two steps, the model's memory consumption reduces significantly, while achieving a competitive accuracy. This strategy also lets us use novel batching techniques that were not feasible with expander graphs of a large degree. Having a regular degree enables using dense matrix multiplication, which is far more efficient with current GPU and TPU devices.

While our method successfully scales to datasets with over two million nodes, it relies on large CPU memory for the attention score estimation for these datasets. For extremely large datasets, this is still infeasible without highly distributed computation. Estimated attention scores can be shared and used for training various networks based on attention scores, however, so this only needs to only be computed once per dataset and depth. An area for potential future work is to combine sampling with simultaneous attention score estimation in a dynamic way, scaling this estimation to very large graphs.

Acknowledgments and Disclosure of Funding

This work was supported in part by the Natural Sciences and Engineering Resource Council of Canada, the Fonds de Recherche du Québec - Nature et technologies (under grant ALLRP-57708-2022), the Canada CIFAR AI Chairs program, the BC DRI Group, Calcul Québec, Compute Ontario, and the Digital Resource Alliance of Canada. Honghao Lin was supported in part by a Simons Investigator Award, NSF CCF-2335412, and a CMU Paul and James Wang Sercomm Presidential Graduate Fellowship.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Achlioptas, D., Karnin, Z. S., and Liberty, E. (2013). Near-optimal entrywise sampling for data matrices. *Advances in Neural Information Processing Systems*, 26.

Chen, J., Gao, K., Li, G., and He, K. (2022a). Nagphormer: Neighborhood aggregation graph transformer for node classification in large graphs. *CoRR*, abs/2206.04910.

Chen, Y., Zeng, Q., Hakkani-Tur, D., Jin, D., Ji, H., and Yang, Y. (2022b). Sketching as a tool for understanding and accelerating self-attention for long sequences. In Carpuat, M., de Marneffe, M., and Ruíz, I. V. M., editors, *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 5187–5199. Association for Computational Linguistics.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. (2021). Rethinking attention with performers. In *ICLR*.

Deng, C., Yue, Z., and Zhang, Z. (2024). Polynormer: Polynomial-expressive graph transformer in linear time. *arXiv preprint arXiv:2403.01232*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Di Giovanni, F., Giusti, L., Barbero, F., Luise, G., Lio, P., and Bronstein, M. M. (2023a). On over-squashing in message passing neural networks: The impact of width, depth, and topology. In *International Conference on Machine Learning*, pages 7865–7885. PMLR.

Di Giovanni, F., Rusch, T. K., Bronstein, M. M., Deac, A., Lackenby, M., Mishra, S., and Veličković, P. (2023b). How does over-squashing affect the power of gnns? *arXiv preprint arXiv:2306.03589*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Dwivedi, V. P. and Bresson, X. (2020). A generalization of transformer networks to graphs. *CoRR*, abs/2012.09699.

Efraimidis, P. S. and Spirakis, P. G. (2006). Weighted random sampling with a reservoir. *Information processing letters*, 97(5):181–185.

Franks, B. J., Morris, C., Velingker, A., and Geerts, F. (2024). Weisfeiler-leman at the margin: When more expressivity matters. *arXiv preprint arXiv:2402.07568*.

Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., and Monti, F. (2020). Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. (2021). OGB-LSC: A large-scale challenge for machine learning on graphs. *CoRR*, abs/2103.09430.

Johnson, W. B. (1984). Extensions of lipshitz mapping into hilbert space. In *Conference modern analysis and probability, 1984*, pages 189–206.

 https://doi.org/10.52202/079017-2277

Kakade, S. and Shakhnarovich, G. (2009). Lecture notes in large scale learning. https://home.ttic.edu/~gregory/courses/LargeScaleLearning/lectures/jl.pdf.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kong, K., Chen, J., Kirchenbauer, J., Ni, R., Bruss, C. B., and Goldstein, T. (2023). Goat: A global transformer on large-scale graphs. In *International Conference on Machine Learning*, pages 17375–17390. PMLR.

Kreuzer, D., Beaini, D., Hamilton, W. L., Létourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. *arXiv preprint arXiv:2106.03893*.

Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., and Qian, W. (2022). Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, pages 13242–13256. PMLR.

Lim, D., Hohne, F., Li, X., Huang, S. L., Gupta, V., Bhalerao, O., and Lim, S. N. (2021). Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902.

Liu, X., Yan, M., Deng, L., Li, G., Ye, X., and Fan, D. (2021). Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica*, 9(2):205–234.

Mialon, G., Chen, D., Selosse, M., and Mairal, J. (2021). Graphit: Encoding graph structure in transformers. *CoRR*, abs/2106.05667.

Müller, L., Galkin, M., Morris, C., and Rampášek, L. (2023). Attending to graph transformers. *arXiv preprint arXiv:2302.04181*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2020). Geom-gcn: Geometric graph convolutional networks. *arXiv preprint arXiv:2002.05287*.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. (2023). A critical look at the evaluation of GNNs under heterophily: Are we really making progress? *arXiv preprint arXiv:2302.11640*.

Prabhu, A., Varma, G., and Namboodiri, A. M. (2018). Deep expander networks: Efficient deep networks from graph theory. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y., editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XIII*, volume 11217 of *Lecture Notes in Computer Science*, pages 20–36. Springer.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515.

Rusch, T. K., Bronstein, M. M., and Mishra, S. (2023). A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*.

Sejdinovic, D., Sriperumbudur, B., Gretton, A., and Fukumizu, K. (2013). Equivalence of distance-based and RKHS-based statistics in hypothesis testing. *The Annals of Statistics*, 41(5):2263 – 2291.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. (2018). Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*.

Shirzad, H., Lin, H., Venkatachalam, B., Velingker, A., Woodruff, D., and Sutherland, D. J. (2024). A theory for compressibility of graph transformers for transductive learning. In *Machine Learning and Compression Workshop at NeurIPS*.

Shirzad, H., Velingker, A., Venkatachalam, B., Sutherland, D. J., and Sinop, A. K. (2023). Exphormer: Sparse transformers for graphs. In *ICML*.

Székely, G. J. and Rizzo, M. L. (2013). Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference*, 143(8):1249–1272.

Takac, L. and Zabovsky, M. (2012). Data analysis in public social networks. In *International scientific conference and international workshop present day trends of innovations*, volume 1.

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

Topping, J., Di Giovanni, F., Chamberlain, B. P., Dong, X., and Bronstein, M. M. (2021). Understanding over-squashing and bottlenecks on graphs via curvature. *arXiv preprint arXiv:2111.14522*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *NeurIPS*, pages 5998–6008.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2018). Graph attention networks. In *ICLR*.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019). Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR.

Wu, Q., Yang, C., Zhao, W., He, Y., Wipf, D., and Yan, J. (2023). Difformer: Scalable (graph) transformers induced by energy constrained diffusion. *arXiv preprint arXiv:2301.09474*.

Wu, Q., Zhao, W., Li, Z., Wipf, D. P., and Yan, J. (2022). Nodeformer: A scalable graph structure learning transformer for node classification. *NeurIPS*, 35:27387–27401.

Wu, Q., Zhao, W., Yang, C., Zhang, H., Nie, F., Jiang, H., Bian, Y., and Yan, J. (2024). Simplifying and empowering transformers for large-graph representations. *Advances in Neural Information Processing Systems*, 36.

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*, abs/2106.05234.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. (2020). Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. K. (2020). Graphsaint: Graph sampling based inductive learning method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

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

Zhao, J., Li, C., Wen, Q., Wang, Y., Liu, Y., Sun, H., Xie, X., and Ye, Y. (2021). Gophormer: Ego-graph transformer for node classification. *CoRR*, abs/2110.13094.

 https://doi.org/10.52202/079017-2277

A Notation Table

Table 4: A summary of the notation used in this paper. The hat notation always refers to a compressed network equivalent of a vector or matrix from the reference network.

Notation	Definition
n	The number of nodes in the graph
m	The number of attention edges in total, including graph and expander edges
d	Hidden dimension of a narrow network
D	Hidden dimension of the original large graph
L	The total number of layers in the network
ℓ	Arbitrary layer index
$\mathbf{V}$	Value mapping of the vectors in the attention mechanism
$\mathbf{Q}$	Query mapping of the vectors in the attention mechanism
$\mathbf{K}$	Key mapping of the vectors in the attention mechanism
$\mathbf{W}_{\cdot}^{(\ell)}$	Weight matrix of mapping such as key, query, value, edge features, or bias in layer ℓ
$\widehat{\mathbf{W}}_{\cdot}^{(\ell)}$	Low dimensional network's weight matrix for a mapping in layer ℓ
$\mathbf{M}_{\cdot}$	A linear mapping matrix (usually from the higher dimension to the smaller)
ReLU	Rectified Linear Unit
$\mathbf{H}^{(\ell)}$	Output of layer $\ell - 1$ from the reference network
$\tilde{\mathbf{H}}^{(\ell)}$	A low-rank estimation of $\mathbf{H}^{(\ell)}$
$\widehat{\mathbf{H}}^{(\ell)}$	Output of layer $\ell - 1$ from a compressed network
$h_i^{(\ell)}$	column i of matrix $\mathbf{H}^{(\ell)}$
$a_{ij}^{(\ell)}$	The Attention score between nodes i and j in layer ℓ
$\hat{a}_{ij}^{(\ell)}$	The attention score between nodes i and j in layer ℓ from a smaller network

B Dataset Descriptions

Below, we provide descriptions of the datasets on which we conduct experiments. A summarized statistics of these datasets have been provided in Table 5.

Amazon datasets Amazon Computers and Amazon Photo are Amazon co-purchase graphs. Nodes represent products purchased. An edge connects a pairs of products purchased together. Node features are bag-of-words encoded reviews of the products. Class labels are the product category.

Amazon2M Amazon2M dataset is a graph from the co-purchasing network. Each node represents an item. Edges between items represents products purchased together. The node features are generated from the product description. The node labels are from the top-level categories the product belongs to.

Actor dataset The actor dataset is created by the actor-only subgraph of a larger graph of actor, director, writer, and film co-occuring on a Wikipedia page, limited to English-language films. Each node corresponds to an actor. Edges denote co-occurence on a Wikipedia page. Node features are based on the terms in the actor's page. The prediction task is categorizing into one of five categories (Pei et al., 2020).

Coauthor datasets The datasets, CS and Physics are co-authorship graphs from Microsoft Academic Graph. The nodes represent the authors and an edge connects two authors who share a paper. The node features are the keywords in the papers. The class represents the active area of study for the author.

ogbn-arxiv (Hu et al., 2021) The ogbn-arxiv dataset is from OGBN datasets. The nodes represents the papers and edges represent the citations between the papers. Nodes are 128-dimensional feature vector that is an average of the embeddings of words in the title and abstract. The prediction task is to identify the category of the 40 subject areas.

ogbn-proteins dataset The ogbn-proteins dataset is an undirected graph with edge weights and types based on species. The nodes represent proteins from eight different species. Edges indicate various biologically meaningful associations between the proteins (e.g., co-expression, homology etc.). The edges are eight-dimensional, with each dimension having a value from [0,1] indicates the confidence score. The prediction task is a multi-label binary classification among 112 labels — to predict the presence of protein functions. The performance measurement is the average of ROC-AUC scores across the 112 tasks.

Minesweeper The dataset is a graph representation of the 100x100 grid from the Minesweeper game. A node represents a cell and the edges connect a node to its eight neighboring cells. 20% of the nodes are marked as mines. The features of the nodes are the one-hot encoding of the mines among the neighbors. For 50% of the nodes the features are unknown and indicated by a separate binary feature.

Tolokers Tolokers is a graph representation of the workers in a crowd-sourcing platform, called Toloka. Each node represents a worker. Two nodes are connected if the workers have worked on the same task. Node features are based on the worker's task performance statistics and other profile information. The task is to predict which nodes have been banned for a project.

Pokec Pokec is a large-scale social network dataset. Nodes represents users of the network. Nodes features include profile data like geographical region, age etc. The task is to predict the gender of users based on the graph.

Table 5: Dataset statistics. The reported number of edges is the number of directed edges, which will be twice the number of actual edges for the undirected graphs.

Dataset	Nodes	Edges	Average Degree	Node Features	Classes	Metric
Amazon Photo	7,487	238,162	31.13	745	8	Accuracy
Coauthor Physics	34,493	495,924	14.38	8,415	5	Accuracy
Amazon Computer	13,381	491,722	35.76	767	10	Accuracy
Coauthor CS	18,333	163,788	8.93	6,805	15	Accuracy
ogbn-arxiv	169,343	2,332,486	13.77	128	40	Accuracy
Actor	7,600	33,391	4.39	932	5	Accuracy
Minesweeper	10,000	78,804	7.88	7	2	AUC
Tolokers	11,758	1,038,000	88.28	10	10	AUC
Pokec	1,632,803	30,622,564	18.75	65	2	AUC
ogbn-proteins	132,534	79,122,504	597.00	8	112	AUC
Amazon2M	2,449,029	123,718,280	50.52	100	47	AUC

C Experiment Details

C.1 Time-Memory Trade-off

One advantage of our method is that the time and memory can be traded without sacrificing the accuracy of the method. Figure 5 shows this trade-off on two datasets ogbn-proteins and arxiv.

C.2 Hyperparameters

In our networks, we use a higher expander degree than what was used in the EXPHORMER paper. Since many of these edges will get a small attention score, a higher attention score increases the receptive field of the nodes, letting the final network be able to sample from wider options and have better access to long-range dependencies. We also noticed, the attention scores in the first layer are usually more flat than the other layers and so we usually sample more edges from the first attention layer. For the comparisons both on the results and the memory we have given the same expander degree to the Exphormer and the ogbn-arxiv dataset could barely fit into a 40GB GPU memory device with higher expander degree. For the attention score estimator network, we do not use dropout, and we only use one attention head in these networks. The number of layers is always equal between both networks.

https://doi.org/10.52202/079017-2277

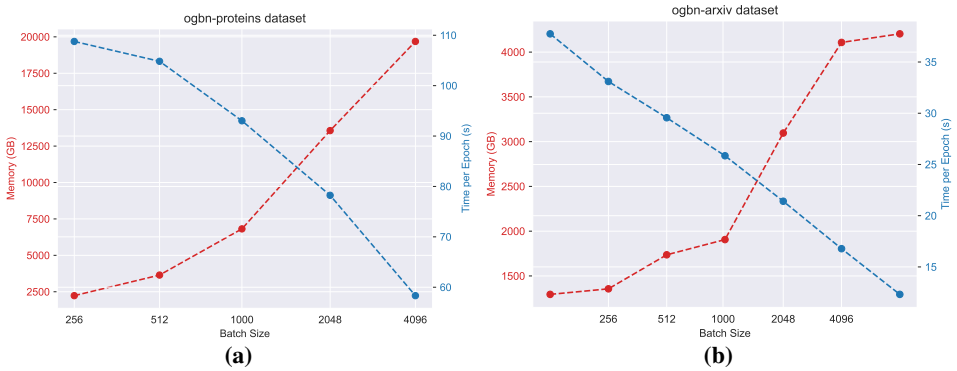

Figure 5: The memory, run-time trade-off for ogbn-proteins and ogbn-arxiv datasets. It is worth mentioning that the experiments with different batch sizes yield similar results for test accuracy/AUC. Memory and time can be traded in our approach.

We use AdamW optimization algorithm in all our networks and use a cosine learning rate scheduler with it. We use weight decay of $1e-3$ in all networks. We use layer norm in attention score estimator networks to keep attention scores more meaningful, but use a batch norm for better results in the final SPEXPHORMER model. Other key hyperparameters can be found in Tables 6,7, and 8.

Table 6: Hyperparameters used for training the networks for homophilous datasets.

Hyperparameter	OGBN-Arxiv	Computer	Photo	CS	Physics
Attention Score Estimator					
L	3	4	4	4	4
d_s	8	4	4	4	4
Num Epochs	200	200	200	200	200
Learning Rate	0.01	0.1	0.001	0.002	0.001
Final Spexphormer Network					
d_l	96	80	56	64	64
$\deg_\ell$	[6, 6, 6]	[5, 5, 5, 5]	[5, 5, 5, 5]	[5, 5, 5, 5]	[5, 5, 5, 5]
Number of Heads	2	2	2	2	2
Learning Rate	0.01	0.001	0.01	0.002	0.001
Num Epochs	600	150	100	120	80
Dropout	0.3	0.5	0.5	0.4	0.4

Table 7: Hyperparameters used for training the networks for heterophilic datasets.

Hyperparameter	Actor	Minesweeper	Tolokers
Attention Score Estimator			
L	3	4	4
d_s	4	4	4
Num Epochs	100	100	200
Learning rate	0.01	0.01	0.01
Final Spexphormer Network			
d_l	32	32	32
$\deg_\ell$	[2, 2, 2]	[12,5,5,5]	[12, 10, 10, 10]
Number of Heads	4	4	4
Learning Rate	0.01	0.01	0.01
Num Epochs	100	80	200
Dropout	0.5	0.2	0.25

Table 8: Hyperparameters used for training the networks for the large graphs datasets.

Hyperparameter	ogbn-proteins	Amazon2M	Pokec
Attention Score Estimator			
L	2	2	2
d_s	8	8	8
expander degree	200	30	30
Num Epochs	150	150	150
Learning rate	0.01	0.01	0.01
Final Spexphormer Network			
d_l	64	128	64
$\deg_\ell$	[50, 30]	[10,10]	[20, 20]
Number of Heads	1	1	1
Learning Rate	0.005	0.001	0.01
Num Epochs	200	200	300
Dropout	0.1	0.2	0.2
Batch size	256	1000	500
GPU Memory	2232MB	3262MB	2128MB

C.3 Hardware

For all trainings of the medium-sized graph datasets and the final network training of the large-sized graphs, we used GPUs of type A100 with 40GB memory, and V100, both 32GB and 16GB versions. While these are powerful GPUs, we have always monitored the GPU memory usage for computational efficiency, ensuring that no more than 8GB is used for whole graph training and no more than 4GB of GPU memory is used with batching. Training with even less memory is feasible with smaller batch sizes.

For calculating the attention scores on the large graph datasets, we have used CPU devices Intel Xeon E5-2680 v4, with 500GB of memory. Except for the Amazon2M dataset, for the other datasets 200GB of memory would be sufficient.

D Theory

In this section, we theoretically analyze the compressibility of the Graph Transformer architecture and also sparsification guarantees using the attention score estimator network.

For simplification, we use the following formulation of a single head Transformer network:

$$h_i^{(\ell+1/2)} = \sum_{j=1}^{deg_i} a_{ij}^{(l)} \mathbf{V}_j^{(\ell)},$$

$$h_i^{(\ell+1)} = \mathbf{W}_2^{(\ell)} \left(\sigma\left(\mathbf{W}_1^{(\ell)}\left(h_i^{(\ell+1/2)}\right)\right)\right),$$

$$a_{ij}^{(l)} = \frac{\exp\left(\mathbf{K}_j^{(\ell)} \cdot \mathbf{Q}_i^{(\ell)}\right)}{\sum_{u \in \mathcal{N}_H(i)} \exp\left(\mathbf{K}_u^{(\ell)} \cdot \mathbf{Q}_i^{(\ell)}\right)},$$

where, $\mathbf{V}^{(\ell)} = \mathbf{W}_V^{(\ell)} h^{(\ell)}$, $\mathbf{Q}^{(\ell)} = \mathbf{W}_Q^{(\ell)} h^{(\ell)}$, $\mathbf{K}^{(\ell)} = \mathbf{W}_K^{(\ell)} h^{(\ell)}$, and σ can be any 1-Lipchitz activation function, such as ReLU, which has been used in practice in our networks. We remove the normalization parts from the architecture but assume that in all steps for all vectors, $\|X_i\|_2, \|h_i^{(\ell+1/2)}\|_2, \|h_i^{(\ell)}\|_2 \leq \sqrt{\alpha}$, and all linear mapping $\mathbf{W}_\cdot$ matrices' operator norm is bounded by a constant β. The first assumption is realistic because of the layer-norm applied between the layers in real-world architectures. The second assumption is also justified as the operator norms are near 2 in the initialization of the network by the default PyTorch initialization and during the optimization

https://doi.org/10.52202/079017-2277

we expect the operator norm to not increase drastically from the initialization. Also, we assume $h^{(0)} = X$, which is the input features. For a simpler notation, we will use D for a hypothetical large network hidden dimension in this analysis, and d is the hidden dimension of the narrow network. For simplicity, in our analysis, we assume $X \in \mathbb{R}^{n \times D}$. In case each node has less than D features, we can concatenate them with zeros.

D.1 On the Compressibility of the Graph Transformer

Our approach uses a narrow network to estimate the attention scores. We want to show if we have a large network with good accuracy for a task on a graph, we can have a less complex network that can work on the same input graph and the error of this network is bounded by $\mathcal{O}(\varepsilon)$ from the large network.

The most memory/time-intensive part of a Transformer architecture is its attention score calculation part. The rest of the sections are node/token-wise and linear with respect to the number of nodes. The attention score estimation part of a full-Transformer layer requires $\mathcal{O}(n^2 d)$ operations and $\mathcal{O}(md)$ operators are required for a sparse Transformer with m attention edges. In the main Exphormer network, this would also be more intensive as the edge features mappings require $\mathcal{O}(md^2)$ operations, but since we replace edge feature mappings with edge embeddings by their type, this part in case we do not have other edge features is $\mathcal{O}(md)$, but m still can be $\omega(n)$, and it will be the most computationally-intensive part.

Assume we have a large network with L layers, where L is $\mathcal{O}(1)$, and hidden dimension D, we will show that there is a similar network with L layers where the attention score calculation matrices $\mathbf{W}_Q, \mathbf{W}_K \in \mathbb{R}^{D \times d}$, and all other matrices are of the same size and d is $\mathcal{O}(C^L \frac{\log n}{\epsilon^2})$, where C is a constant based on α and β. For this proof we use the distributional Johnson-Lindenstrauss transform lemma (Johnson, 1984):

Lemma D.1 (Johnson-Lindenstrauss Transform Lemma (*JLT*))**.** *Assume $0 < \epsilon, \delta < \frac{1}{2}$ and any positive integer D, if $d = \mathcal{O}(\frac{\log(1/\delta)}{\epsilon^2})$, there exist a distribution over matrices $\mathbf{M} \in \mathbb{R}^{d \times D}$ that for any $x \in \mathbb{R}^D$ and $\|x\| = 1$:*

$$\Pr(\|\mathbf{M}x\| - 1 > \epsilon) < \delta$$

The following corollary is an immediate conclusion from the *JLT*.

Corollary D.2. *Assume $0 < \epsilon, \delta < \frac{1}{2}$ and any positive integer D, if $d = \mathcal{O}(\frac{\log(1/\delta)}{\varepsilon^2})$, there exist a distribution over matrices $\mathbf{M} \in \mathbb{R}^{d \times D}$ that for any $x, y \in \mathbb{R}^D$:*

$$\Pr((1-\varepsilon)\|x-y\| < \|\mathbf{M}x - \mathbf{M}y\| < (1+\varepsilon)\|x-y\|) < \delta$$

This can derived by replacing x from *JLT* with $\frac{x-y}{\|x-y\|}$.

From this, we can derive another corollary about the dot product of the vectors in low-dimensional space.

Corollary D.3 (JLT-dot product)**.** *Assume $0 < \epsilon, \delta < \frac{1}{2}$ and any positive integer D, if $d = \mathcal{O}(\frac{\log(1/\delta)}{\varepsilon^2})$, there exist a distribution over matrices $\mathbf{M} \in \mathbb{R}^{d \times D}$ that for any $x, y \in \mathbb{R}^D$, and $\|x\|, \|y\| \leq \sqrt{\alpha}$:*

$$\Pr((1-\varepsilon\alpha)x^\mathsf{T}y < x^\mathsf{T}\mathbf{M}^\mathsf{T}\mathbf{M}y < (1+\varepsilon\alpha)x^\mathsf{T}y) < \delta$$

For the proof see (Kakade and Shakhnarovich, 2009, Corollary 2.1). As a result of this corollary, if we have m pairs of vectors (x_i, y_i), and for each i $\|x_i\|_2, \|y_i\|_2 \leq \sqrt{\alpha}$ of $\sqrt{\alpha}$, and $d = \mathcal{O}(\frac{\log(m)}{\varepsilon^2})$, there exists an $\mathbf{M}$ such that for all these pairs $|x_i^\mathsf{T}\mathbf{M}^\mathsf{T}\mathbf{M}y_i - x_i^\mathsf{T}y_i| < \varepsilon\alpha$. The proof can be done using a union bound over the error from Corollary D.3. Also, in our case where m is the number of edges, we know that $m \leq n^2$, thus we can also say $d = \mathcal{O}(\frac{\log(n)}{\varepsilon^2})$.

Theorem D.4. *Assume we have a Transformer network $\mathcal{T}$ with arbitrary large hidden dimension D, $L = O(1)$ layers, and in this network, in all layers, we have $\|h_\cdot\|_2 \leq \sqrt{\alpha}$, and $\|\mathbf{W}_\cdot\|_{op} \leq \beta$.*

There exists a Transformer $\widehat{\mathcal{T}}$, that for any layer $\mathbf{W}_Q$ and $\mathbf{W}_K$ are in $\mathbb{R}^{d\times D}$ for a $d = \mathcal{O}(\frac{\log n}{\varepsilon^2})$, with a sufficiently small ε, and for all $i \in [n]$, $\|\mathcal{T}(X)_i - \widehat{\mathcal{T}}(X)_i\|_2 = \mathcal{O}(\varepsilon)$. And furthermore, for any attention score $\frac{a_{ij}^{(\ell)}}{\widehat{a}_{ij}^{(\ell)}} = 1 + \mathcal{O}(\varepsilon)$.

Proof. In the proof we use hat notation, $\widehat{\square}$, for the vectors and matrices from $\widehat{\mathcal{T}}$, for example, $\hat{h}^{(\ell)}$ are the outputs of layer ℓ, and $\widehat{\mathbf{W}}_\cdot$ are the weight matrices for this network. In all layers for both networks $\mathbf{W}_V, \mathbf{W}_1$, and $\mathbf{W}_2$, are of the same size, so we set $\widehat{\mathbf{W}}_V = \mathbf{W}_V$, $\widehat{\mathbf{W}}_1 = \mathbf{W}_1$, and $\widehat{\mathbf{W}}_2 = \mathbf{W}_2$.

For the proof, we want to find $\varepsilon^{(0)}, \cdots, \varepsilon^{(L)}$ in a way that for any v in layer ℓ, $|h_v^{(\ell)} - \hat{h}_v^{(\ell)}| < \varepsilon^{(\ell)}$. We will find these bounds inductively, starting from the first layer. We have $\varepsilon^{(0)} = 0$, as both networks have the same input, and we want to bound $\varepsilon^{(\ell+1)}$ based on $\varepsilon^{(\ell)}$.

We have $\mathbf{Q}^{(\ell)} = \mathbf{W}_Q^{(\ell)}\mathbf{H}^{(\ell)}$, $\mathbf{K}^{(\ell)} = \mathbf{W}_K^{(\ell)}\mathbf{H}^{(\ell)}$ and assume $\bar{\mathbf{Q}}^{(\ell)} = \mathbf{W}_Q^{(\ell)}\widehat{\mathbf{H}}^{(\ell)}$, $\bar{\mathbf{K}}^{(\ell)} = \mathbf{W}_K^{(\ell)}\widehat{\mathbf{H}}^{(\ell)}$. Because of the operator norm of matrices $\mathbf{W}_Q$ and $\mathbf{W}_K$, for each i we have $\|q_i^{(\ell)} - \bar{q}_i^{(\ell)}\| \le \varepsilon^{(\ell)}\beta$ and $\|k_i^{(\ell)} - \bar{k}_i^{(\ell)}\| \le \varepsilon^{(\ell)}\beta$. Also, we have $\|q_i^{(\ell)}\|, \|k_i^{(\ell)}\| \le \beta\sqrt{\alpha}$, thus $\|\bar{q}_i^{(\ell)}\|, \|\bar{k}_i^{(\ell)}\| \le \beta(\varepsilon^{(\ell)} + \sqrt{\alpha})$. Now, for each pair of i and j, we have:

$$\begin{aligned}
|q_i^{(\ell)} \cdot k_j^{(\ell)} - \bar{q}_i^{(\ell)} \cdot \bar{k}_j^{(\ell)}| &= |q_i^{(\ell)} \cdot k_j^{(\ell)} - \bar{q}_i^{(\ell)} \cdot k_j^{(\ell)} + \bar{q}_i^{(\ell)} \cdot k_j^{(\ell)} - \bar{q}_i^{(\ell)} \cdot \bar{k}_j^{(\ell)}| \\
&\le |q_i^{(\ell)} \cdot k_j^{(\ell)} - \bar{q}_i^{(\ell)} \cdot k_j^{(\ell)}| + |\bar{q}_i^{(\ell)} \cdot k_j^{(\ell)} - \bar{q}_i^{(\ell)} \cdot \bar{k}_j^{(\ell)}| \\
&= |(q_i^{(\ell)} - \bar{q}_i^{(\ell)}) \cdot k_j^{(\ell)}| + |\bar{q}_i^{(\ell)} \cdot (k_j^{(\ell)} - \bar{k}_j^{(\ell)})| \\
&\le \|q_i^{(\ell)} - \bar{q}_i^{(\ell)}\|\|k_j^{(\ell)}\| + \|\bar{q}_i^{(\ell)}\|\|k_j^{(\ell)} - \bar{k}_j^{(\ell)}\| \\
&\le \sqrt{\alpha}\beta\varepsilon^{(\ell)} + (\sqrt{\alpha} + \beta\varepsilon^{(\ell)})\beta\varepsilon^{(\ell)} \\
&= 2\sqrt{\alpha}\beta\varepsilon^{(\ell)} + (\beta\varepsilon^{(\ell)})^2
\end{aligned}$$

On the other hand, according to the D.3, for a $0 < \varepsilon < 1/2$ and $d = \mathcal{O}(\frac{\log(n)}{\varepsilon^2})$ there exists a matrix $\mathbf{M}_{QK} \in \mathbb{R}^{d\times D}$, such that if we define $\widehat{\mathbf{Q}}^{(\ell)} = \mathbf{M}_{QK}\bar{\mathbf{Q}}^{(\ell)}$ and $\widehat{\mathbf{K}}^{(\ell)} = \mathbf{M}_{QK}\bar{\mathbf{K}}^{(\ell)}$, $|\bar{q}_i^{(\ell)} \cdot \bar{k}_j^{(\ell)} - \hat{q}_i^{(\ell)} \cdot \hat{k}_j^{(\ell)}| < \beta^2(\alpha + (\varepsilon^{(\ell)})^2 + 2\sqrt{\alpha}\varepsilon^{(\ell)})\varepsilon$ for all (i, j) pairs in the attention pattern. Note that we can define $\widehat{\mathbf{W}}_Q^{(\ell)} = \mathbf{M}_{QK}^{(\ell)}\mathbf{W}_Q^{(\ell)}$, and $\widehat{\mathbf{W}}_K^{(\ell)} = \mathbf{M}_{QK}^{(\ell)}\mathbf{W}_K^{(\ell)}$, both in $\mathbb{R}^{d\times D}$, as weights for the narrow attention score estimator network. With a triangle inequality we have

$$|q_i^{(\ell)} \cdot k_i^{(\ell)} - \hat{q}_i^{(\ell)} \cdot \hat{k}_i^{(\ell)}| < \beta^2(\alpha + (\varepsilon^{(\ell)})^2 + 2\sqrt{\alpha}\varepsilon^{(\ell)})\varepsilon + 2\sqrt{\alpha}\beta\varepsilon^{(\ell)} + (\beta\varepsilon^{(\ell)})^2.$$

By setting $\varepsilon^{(\ell)} \le 1$, we have

$$|q_i^{(\ell)} \cdot k_i^{(\ell)} - \hat{q}_i^{(\ell)} \cdot \hat{k}_i^{(\ell)}| < \beta^2(\alpha + 1 + 2\sqrt{\alpha})\varepsilon + \beta(2\sqrt{\alpha} + \beta)\varepsilon^{(\ell)}.$$

Let us define $\varepsilon_a = \beta^2(\alpha + 1 + 2\sqrt{\alpha})\varepsilon + \beta(2\sqrt{\alpha} + \beta)\varepsilon^{(\ell)}$, we have:

$$\begin{aligned}
\widehat{a}_{ij}^{(\ell)} &= \frac{\exp(\hat{q}_i^{(\ell)} \cdot \hat{k}_j^{(\ell)})}{\sum_{u\in\mathcal{N}_H(i)} \exp(\hat{q}_i^{(\ell)} \cdot \hat{k}_u^{(\ell)})} \le \frac{\exp(q_i^{(\ell)} \cdot k_j^{(\ell)} + \varepsilon_a)}{\sum_{u\in\mathcal{N}_H(i)} \exp(q_i^{(\ell)} \cdot k_j^{(\ell)} - \varepsilon_a)} \le a_{ij}^{(\ell)} \exp(2\varepsilon_a) \\
\widehat{a}_{ij}^{(\ell)} &= \frac{\exp(\hat{q}_i^{(\ell)} \cdot \hat{k}_j^{(\ell)})}{\sum_{u\in\mathcal{N}_H(i)} \exp(\hat{q}_i^{(\ell)} \cdot \hat{k}_u^{(\ell)})} \ge \frac{\exp(q_i^{(\ell)} \cdot k_j^{(\ell)} - \varepsilon_a)}{\sum_{u\in\mathcal{N}_H(i)} \exp(q_i^{(\ell)} \cdot k_u^{(\ell)} + \varepsilon_a)} \ge a_{ij}^{(\ell)} \exp(-2\varepsilon_a)
\end{aligned}$$

Now we bound $\|h_i^{(\ell+1/2)} - \hat{h}_i^{(\ell+1/2)}\|$:

https://doi.org/10.52202/079017-2277

$$
\begin{aligned}
\|h_i^{(\ell+1/2)} - \hat{h}_i^{(\ell+1/2)}\| &= \| \sum_{j \in Nei(i)} a_{ij}^{(\ell)} v_j^{(\ell)} - \widehat{a}_{ij} \hat{v}_j^{(\ell+1/2)} \| \\
&= \| \sum_{j \in Nei(i)} a_{ij}^{(\ell)} v_j^{(\ell)} - \widehat{a}_{ij}^{(\ell)} v_j^{(\ell)} + \widehat{a}_{ij}^{(\ell)} v_j^{(\ell)} - \widehat{a}_{ij} \hat{v}_j^{(\ell)} \| \\
&= \| \sum_{j \in Nei(i)} (a_{ij}^{(\ell)} - \widehat{a}_{ij}^{(\ell)}) v_j^{(\ell)} + \widehat{a}_{ij}^{(\ell)} (v_j^{(\ell)} - \hat{v}_j^{(\ell)}) \| \\
&= \|(v_j^{(\ell)} - \hat{v}_j^{(\ell)}) + v_j^{(\ell)} \sum_{j \in Nei(i)} (a_{ij}^{(\ell)} - \widehat{a}_{ij}^{(\ell)}) \| \\
&\leq \|v_j^{(\ell)} - \hat{v}_j^{(\ell)}\| + \|v_j^{(\ell)}\| \sum |a_{ij}^{(\ell)} - \widehat{a}_{ij}^{(\ell)}| \\
&\leq \varepsilon^{(\ell)} \beta + \sqrt{\alpha} \sum \max(1 - \exp(-2\varepsilon_a), \exp(2\varepsilon_a) - 1) a_{ij}^{(\ell)} \\
&\leq \varepsilon^{(\ell)} \beta + \sqrt{\alpha}(\exp(2\varepsilon_a) - 1),
\end{aligned}
$$

and since $1 + x < \exp(x) < 1 + 2x$ for $0 < x < 1$, if we have $\varepsilon_a < 1$, we have

$$
\|h_i^{(\ell+1/2)} - \hat{h}_i^{(\ell+1/2)}\| \leq \beta \varepsilon^{(\ell)} + 4\sqrt{\alpha} \varepsilon_a \tag{1}
$$

For the feed-forward network part, we know that this network is β^2-Lipschitz because $\mathbf{W}_1^{(\ell)}$ and $\mathbf{W}_2^{(\ell)}$ have maximum operator norm β and σ is a 1-Lipschitz activation function. Thus we have

$$
\|h_i^{(\ell+1)} - \hat{h}_i^{(\ell+1)}\| \leq \beta^2(\beta \varepsilon^{(\ell)} + 4\sqrt{\alpha}\varepsilon_a) = (\beta^3 + 8\beta\alpha + 4\beta^2\sqrt{\alpha})\varepsilon^{(\ell)} + 4\beta^2(\alpha\sqrt{\alpha} + 2\alpha + \sqrt{\alpha})\varepsilon.
$$

Both $\beta^3 + 8\beta\alpha + 4\beta^2\sqrt{\alpha}$ and $4\beta^2(\alpha\sqrt{\alpha} + 2\alpha + \sqrt{\alpha})$ are constants, and if we define them as c_1 and c_2, we have

$$
\varepsilon^{(\ell+1)} \leq c_1 \varepsilon^{(\ell)} + c_2 \varepsilon
$$

Given $\varepsilon^{(0)} = 0$, as both networks get the same input, we have

$$
\begin{aligned}
\varepsilon^{(L)} &\leq c_1 \varepsilon^{(L-1)} + c_2 \varepsilon \\
&\leq c_1(c_1 \varepsilon^{(L-2)} + c_2 \varepsilon) + c_2 \varepsilon \\
&\cdots \\
&\leq c_2 \varepsilon (c_1^{L-1} + \cdots + c_1) \\
&= \frac{c_1(c_2^L - 1)}{c_2 - 1} \varepsilon
\end{aligned}
$$

While the error increases exponentially with the number of layers, when we have $L = O(1)$, then the error is bounded by a constant factor of chosen ε. Now, we know that $\|\mathcal{T}(X)_i - \widehat{\mathcal{T}}(X)_i\|_2 \leq \varepsilon^{(L)} = \mathcal{O}(\varepsilon)$.

□

While from the theorem it seems that the error is increasing exponentially by the layers, in practice the maximum number of layers used in this work is four with most large graph experiments using just two layers. Thus the constant factor will not be as large as it might look. Also, in real-world graphs usually, the columns of X are not quite n distinct vectors and many vectors would be equal or very similar to each other if we have κ unique vectors in the first layer the complexity for the d can be reduced to $\mathcal{O}(\frac{\log \kappa}{\varepsilon^2})$. In the homophily graphs the representations $h^{(\ell)}$ tend to converge to each other and thus again the number of unique vectors will be reduced letting us have smaller d, but these assumptions are not considered in the proof as we keep it general.

Although we have proved the existence of the $\widehat{\mathcal{T}}$, this does not mean that training with a gradient-based algorithm will necessarily lead to the introduced weights, but this gives at least the guarantee

that such a network exists. However, on the other hand, it is also possible that the training process finds a set of weights that work better than the weights constructed in this proof.

Theorem D.4 by narrowing the attention score calculation part reduced the complexity from $\mathcal{O}(mD + nD^2)$ to $\mathcal{O}(md + nD^2)$, and for dense graphs or in scenarios we add denser expander graphs, where $m \gg n$, already the introduced network has a much lower complexity. However, our narrow network uses narrow hidden dimensions in all steps and has complexity $\mathcal{O}(md + nd^2)$. Proving the same guarantee along the whole network is not easy, if not impossible, without any further assumptions on X and the large network.

D.2 Analysis of the Sampling Process

After training a network with a smaller width d, we sample the edges from the original graph and use them in the second-phase training with a large hidden width D. In this section, we shall analyze our sampling process. Formally, we model our process as follows. Suppose that A is the attention score matrix with hidden width D, then we sample and rescale s entries of A to form a sparse matrix B where the goal is the matrix B can approximate A well, i.e., $\|A - B\|_2 \leq \varepsilon \|A\|_2$. However, recall that we can not access the entries of A precisely. Instead, we consider another attention score matrix A', which corresponds to hidden width d.

The first question is how many samples we indeed need to form the matrix B that approximates A well? To answer this, we have the following lemma for the attention score matrix A.

Theorem D.5. *Suppose that an $n \times n$ matrix A satisfies the following condition.*

1. *For each i, we have $\|A_{(i)}\|_1 = 1$*
2. *$\max_j \|A^{(j)}\|_1 = K$*
3. *Each column $A^{(j)}$ is ℓ-sparse*

Then, consider the sampling procedure that samples $s \geq s_0 = \mathcal{O}(nK \log n/(\varepsilon^2 \|A\|_2^2)) = \mathcal{O}(n\ell \log n/(\varepsilon^2 K))$ entries of A with replacement

1. *For each sample B_t, the probability that B_t samples entry A_{ij} is $p_{ij} = \frac{1}{n} \cdot \frac{|A_{ij}|}{\|A_{(i)}\|_1} = \frac{1}{n}|A_{ij}|$ (with a rescale factor $1/p_{ij}$, i.e., $B_t[i,j] = A_{ij}/p_{ij}$), and each B_t only sample one entry of A.*
2. *Form the matrix $B = (B_1 + B_2 + \cdots + B_s)/s$*

Then, we have that with probability at least $9/10$,

$$\|A - B\|_2 \leq \varepsilon \|A\|_2.$$

To prove this lemma, we need the following Matrix Bersiten inequality.

Lemma D.6 (Matrix Bersiten inequality)**.** *Consider a finite sequence X_i of i.i.d. random $m \times n$, where $\mathbb{E}[X_i] = 0$ and $\|X_i\|_2 \leq R$. Let $\sigma^2 = \max\{\|\mathbb{E}[X_i X_i^T]\|_2, \|\mathbb{E}[X_i^T X_i]\|_2\}$. For some fixed $s \geq 1$, Let $= (X_1 + X_2 + \cdots + X_s)/s$, then we have that*

$$\Pr[\|X\|_2 \geq \varepsilon] \leq (m+n) \cdot \exp\left(\frac{s\varepsilon^2}{-\sigma^2 + R\varepsilon/3}\right)$$

Proof. We follow a similar proof strategy in Achlioptas et al. (2013). At a high level, the work of Achlioptas et al. (2013) considers the matrix Bernstein inequality where we have the tail bound

$$\Pr[\|A - B\|_2 \geq \varepsilon] \leq 2n \cdot \exp\left(\frac{s\varepsilon^2}{\sigma^2 + R\sigma/3}\right)$$

is dependent on the following two quantities

$$\sigma^2 = \max\{\|\mathbb{E}[(A - B_1)(A - B_1)^T]\|, \|\mathbb{E}[(A - B_1)^T (A - B_1)]\|\}$$
$$R = \max\|A - B_1\| \text{ over all possible realizations of } B_1 .$$

https://doi.org/10.52202/079017-2277

Where B_1 is the matrix that only samples one entry and the final output $B = (B_1 + B_2 + \cdots + B_s)/s$. Then, instead we consider the following quantity,

$$\tilde{\sigma}^2 = \max\left\{ \max_i \sum_j A_{ij}^2/p_{ij}, \max_j \sum_i A_{ij}^2/p_{ij} \right\}$$

$$\tilde{R} = \max_{ij} |A_{ij}|/p_{ij} \ .$$

It is shown in Lemma A.2 of Achlioptas et al. (2013) we have that $|\sigma/\tilde{\sigma} - 1| \leq \frac{\|A\|_2^2}{\sum_i \|A_{(i)}\|_1^2}$ and $|R/\tilde{R} - 1| \leq \frac{\|A\|_2}{\|A\|_1}$ and from the condition of the matrix A we have both of the upper bounds are at most 1. Hence, we only need to consider $\tilde{\sigma}$ and $\tilde{R}$. Back to our case, we have that $p_{ij} = \frac{1}{n} \cdot \frac{|A_{ij}|}{\|A_{(1)}\|_1} = \frac{1}{n} \cdot |A_{ij}|$, from this and the assumption of A we have

$$\tilde{\sigma}^2 = n \cdot \max\left\{ \max_i \sum_j |A_{ij}|, \max_j \sum_i |A_{ij}| \right\} \leq n \cdot K$$

$$\tilde{R} = \max_{ij} |A_{ij}|/p_{ij} = n \ .$$

Hence, to make $\delta \leq 0.1$, we only need to set $\varepsilon' = \varepsilon \|A\|_2$ in the Matrix Bernstein inequality and then we have $s \geq O(nK \log n/(\varepsilon^2 \|A\|_2^2))$. Finally, note that if $\|A^{(j)}\|_1 = K$, then we have $\|A\|_2 \geq \|Ae_j\|_2 = \|A^{(j)}\|_2 \geq K/\sqrt{\ell}$, which means that $nK \log n/(\varepsilon^2 \|A\|_2^2) \leq n\ell \log n/(\varepsilon^2 K)$. □

However, as mentioned, we can not access the value of the entries of A but the entries of A' (which corresponds to the trained network with a small hidden width d). We next show that even in the case where we sample the entries of A from A', we can still get the same order of the bound if the entries of A are not under-estimated seriously in A'.

Proposition D.7. *Suppose that the matrices A and A' satisfy the condition in Theorem D.5 and for every i, j we have*

$$|A'_{ij}| \geq \frac{1}{\alpha} |A_{ij}|$$

for some sufficiently large constant α. Then consider the same sampling procedure in Theorem D.5 but sampling the entries of A from the value of A'. Then, the guarantee in Theorem D.5 still holds.

Proof. We only need to note that from the assumption, the actual sampling probability $p'_{ij} \geq \frac{1}{\alpha} \cdot p_{ij}$ in Theorem D.5, hence it will increase the $\tilde{\sigma}^2$ and $\tilde{R}$ by at most α times, which means that we can increase s by an α factor to make the error probability at most 0.1. □

E Discussion

Graph datasets arise from various domains, meaning that they might have differing inductive biases. More expressive methods may not necessarily yield better results on all datasets (Franks et al., 2024). Depending on the architecture and the task, more complex models can even lead to poorer results. Here, we discuss possible scenarios in which our model can be a good fit as well as the shortcomings of other classes of models that are overcome by our model.

Graph Structure The relevance of the structure of the graph to the task can vary. For the simple synthetic task introduced in 1, the structure of the graph does not matter. So Transformers without inductive biases of the graphs are expressive enough to solve this problem; however message-passing networks will be restricted to the graph edges and rely on enough number of layers and may be challenged by oversquashing and oversmoothing problems. On the other hand, if the structure of the graph matters, such as counting the number of neighbor nodes with the same color for each node, the structure and the edges will be an important part. Transformers without expressive enough encodings to identify the graph edges will fail in this task. On the other hand, MPNNs even with one layer can easily solve this problem. Our approach enables solving problems in either case, by having both

expander graphs for universal information propagation and the actual graph edges for inductive bias, allowing the model to decide the subset of edges that suit the task better — only graph edges, only expander edges or a combination of both.

Short-range Vs. Long-range Dependencies If the neighboring nodes tend to be from the same class, i.e., *high homophily*, MPNNs and methods such as NAGphormer (Chen et al., 2022a), which summarize the neighborhood have good inductive biases; whereas Transformers without proper identification for the neighborhoods may not be as fit for this task. Heterophily may not necessarily mean long-range dependencies, label of each node may just depend on the neighbor nodes, but still label of the neighbor nodes may be different most of the time. For example, for finding the grammatical function of the words in a sentence from a very long text, neighboring words are usually enough for this identification, and nearby words would be from different classes. On the other hand, some tasks may require long-range dependencies — identifying if there are other people in a social network with similar interests or the synthetic task introduced in 1 are some examples. Local models such as MPNNs would require deeper networks for modeling long-range dependencies that makes them prone to common problems such as oversquashing and oversmoothing (Topping et al., 2021; Di Giovanni et al., 2023b,a; Rusch et al., 2023). Our approach can be reduced to MPNN by giving lower attention scores to the expander edges, for learning on the tasks with short-range dependencies only. And also lets the long-range dependency modeling using expander edges. While models designed specifically for some of these tasks may have the advantage of reduced complexity. But our approach lets learning without concern about the nature of the problem or having domain knowledge for the task or graph.

Subsampling Graphs Many approaches break the graph into sections or subsample nodes or neighbors for training. This approach has shown promising results in many works such as (Zeng et al., 2020; Hamilton et al., 2017; Liu et al., 2021). However, there are many cases in which these approaches are not expressive enough. Clustering the nodes or batching and subsampling based on the neighborhood will not have the required inductive biases to solve the tasks with long-range dependencies. Approaches such as neighbor sampling or connected-subgraph sampling not only inherit the limits of the MPNN networks, but may even miss short-range dependencies. For example, Example (c) in 1 by merely random selection of the neighbors or subgraphs without considering the task. Random subset of node selection that has been used in several promising papers such as Wu et al. (2022, 2023, 2024) gives a chance for nodes from the same label to appear in the same batch, but the batch-size should increase with the graph size accordingly. Very small ratio of batch size to graph size would mean many edges or possible pair of nodes will never be appear in any batch and depending on the task this can limit the power of these models. Also, these models are usually not memory efficient, as graph size grows, they can not keep the batches small, and the required memory grows accordingly. On the other hand, our approach (1) makes smarter selection of neighbors based on the small network's attention scores; (2) our sampling allows making k-hop neighborhood subgraphs from the extended graph connectivity, and (3) allows the training by trading off memory and time, without critical harm to the model's expressive power. Unline the GraphSAGE and SGFormer, which use the full graph for the inference time our model uses the same sampling and batching techniques, letting efficient inference beside the efficient training.

https://doi.org/10.52202/079017-2277

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification:

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

https://doi.org/10.52202/079017-2277

Answer: [Yes]

Justification:

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: No specific societal impacts as opposed to other graph neural network models are expected.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

https://doi.org/10.52202/079017-2277

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our model does not fit in high-risk or misusable models.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

https://doi.org/10.52202/079017-2277