Hierarchical Transformer for Scalable Graph Learning

Wenhao Zhu[1] , **Tianyu Wen**[2] , **Guojie Song**[1] , **Xiaojun Ma**[3] and **Liang Wang**[4]
[1]National Key Laboratory of General Artificial Intelligence, School of Intelligence Science and Technology, Peking University
[2]Yuanpei College, Peking University
[3]Microsoft
[4]Alibaba Group
{wenhaozhu, tianyuwen, gjsong}@pku.edu.cn, xiaojunma@microsoft.com, liangbo.wl@alibaba-inc.com

Abstract

Graph Transformer is gaining increasing attention in the field of machine learning and has demonstrated state-of-the-art performance on benchmarks for graph representation learning. However, as current implementations of Graph Transformer primarily focus on learning representations of small-scale graphs, the quadratic complexity of the global self-attention mechanism presents a challenge for full-batch training when applied to larger graphs. Additionally, conventional sampling-based methods fail to capture necessary high-level contextual information, resulting in a significant loss of performance. In this paper, we introduce the Hierarchical Scalable Graph Transformer (HSGT) as a solution to these challenges. HSGT successfully scales the Transformer architecture to node representation learning tasks on large-scale graphs, while maintaining high performance. By utilizing graph hierarchies constructed through coarsening techniques, HSGT efficiently updates and stores multi-scale information in node embeddings at different levels. Together with sampling-based training methods, HSGT effectively captures and aggregates multi-level information on the hierarchical graph using only Transformer blocks. Empirical evaluations demonstrate that HSGT achieves state-of-the-art performance on large-scale benchmarks with graphs containing millions of nodes with high efficiency.

1 Introduction

Transformer [Vaswani *et al.*, 2017] is now the prevalent universal neural architecture in natural language processing and computer vision with its powerful, lowly inductive-biased self-attention mechanism. The great success of Transformer has encouraged researchers to explore its adaptation to graph machine learning on node-level and graph-level tasks [Ying *et al.*, 2021; Kreuzer *et al.*, 2021; Chen *et al.*, 2022]. While GNNs are known to suffer from inherent limitations in the message-passing paradigm like *over-smoothing* and *neighbor explosion*, the promising performance of these graph Transformer methods has encouraged researchers to expand the Transformer architecture to more scenarios.

Still, challenges arise when scaling Transformer to large graphs. In previous methods, self-attention calculates all pairwise interactions in a graph, indicating that it has quadratic complexity to the total number of nodes. Thus, to perform training on graphs of millions of nodes without substantial modification to the Transformer architecture, one must sample a properly sized subgraph at every batch so that the computational graph can be fit into GPU memory. Using sampling strategies like neighbor sampling in GraphSAGE [Hamilton *et al.*, 2017], we can build a simple scalable graph Transformer model by directly applying existing models like Graphormer on the sampled subgraph. However, there is an intrinsic weakness in this straightforward combination of Transformer architecture and sampling-based training methods. It has been widely observed that high-level context information characterized by global receptive field of self-attention module greatly contributes to Transformer's outstanding performance [Vaswani *et al.*, 2017; Ying *et al.*, 2021]. Considering that the entire input graph is usually far larger than every sampled subgraph, when the receptive field of each node is restricted to the sampled local context, the model may ignore high-level context information, leading to possible performance loss. Meanwhile, if we add globally sampled nodes to the sampled set to reduce context locality, it is likely to introduce much redundant noise because most long-distance neighbors are irrelevant in large graphs, which is further confirmed by our experiments.

In this paper, to alleviate the problem above and find the real potentials of Transformer architecture on large-scale graph learning tasks, we propose HSGT, a hierarchical scalable graph Transformer framework. Our key insight is that, by building graph hierarchies with topological coarsening methods, high-level context information can be efficiently stored and updated with a fused representation of high-level node embeddings. As illustrated in Figure 1, through attention interaction with nodes at higher hierarchical layers, the receptive field of each node is expanded to a much higher scale, making it possible for HSGT to effectively capture high-level structural knowledge in the graph during sampling-based training. Since the number of high-level nodes is

Please refer to https://arxiv.org/abs/2305.02866 for an extended version of this paper.

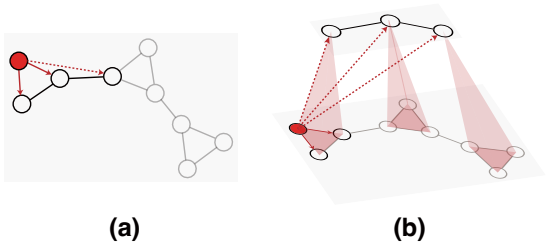

Figure 1: (a) During straightforward sampling-based training, receptive field of each node is restricted to the sampled context, leading to the loss of global context information. (b) In our method, through attention interaction with nodes at higher hierarchies, the receptive field of each node is expanded to a higher scale (red shadows), making it possible for the model to capture high-level knowledge.

marginal compared to the size of original graph, our approach is efficient and brings low extra computational cost. Besides, using adaptively aggregated representation to characterize high-level context information, our method is robust to random noise from long-range irrelevant neighbors.

More concretely, our proposed HSGT architecture utilizes three types of Transformer blocks to support context transformations at different scales. At every hierarchical layer, the horizontal blocks are first performed to exchange and transform information in the local context, then we use vertical blocks to aggregate representation in every substructure and create embeddings for nodes at the higher level. Eventually, readout blocks obtain final node representation by fusing multi-level node embeddings. To achieve scalable training, we have developed a hierarchical sampling method to sample multi-level batches for training and inference, and utilized the historical embedding technique [Fey *et al.*, 2021] to remove inter-batch dependencies and prune the computational graph. Being completely Transformer-based, the resulting HSGT architecture is highly scalable and generalizable, achieving state-of-the-art results on a wide range of datasets from the standard Cora [Sen *et al.*, 2008] to ogbn-products [Chiang *et al.*, 2019] with millions of nodes, outperforming the standard scalable GNN and Transformer baselines. We summarize our main contributions as follows:

- We propose HSGT, a new graph Transformer architecture that efficiently generates high-quality representations for graphs of varying sizes via effective usage of hierarchical structure and multi-level network design.
- We develop the novel hierarchical sampling strategies and apply the historical embedding method, which allow HSGT to be trained efficiently large-scale graphs and gain high performance.
- Extensive experiments show that HSGT achieves state-of-the-art performance against baseline methods on large-scale graph benchmarks with computational costs similar to the standard GraphSAGE method.

2 Related Work

2.1 Graph Transformers

Along with the recent surge of Transformer, many prior works have attempted to bring Transformer architecture to the graph domain, including GT [Dwivedi and Bresson, 2020], GROVER [Rong *et al.*, 2020], Graphormer [Ying *et al.*, 2021], SAN [Kreuzer *et al.*, 2021], SAT [Chen *et al.*, 2022], ANS-GT [Zhang *et al.*, 2022], GraphGPS [Rampášek *et al.*, 2022] and NodeFormer [Wu *et al.*, 2022]. Graphormer [Ying *et al.*, 2021] proposes an enhanced Transformer with centrality, spatial and edge encodings, and achieves state-of-the-art performance on many molecular graph representation learning benchmarks. SAN [Kreuzer *et al.*, 2021] presents a learned positional encoding that cooperates with full Laplacian spectrum to learn the position of each node in the graph. Gophormer [Zhao *et al.*, 2021] applies structural-enhanced Transformer to sampled ego-graphs to improve node classification performance and scalability. SAT [Chen *et al.*, 2022] studies the question of how to encode structural information to Transformers and proposes the Structure-Aware-Transformer to generate position-aware information for graph data. ANS-GT [Zhang *et al.*, 2022] proposes an adaptive sampling strategy to effectively scale up graph Transformer to large graphs.

2.2 Scalable Graph Learning

Modeling complex, real-world graphs with large scale require scalable graph neural models. On large graphs, message-passing GNNs mainly suffer from the *neighbor explosion* phenomenon, since the neighborhood dependency of nodes grows exponentially as the model depth increases, which results in the excessive expansion of computational graphs. Sampling-based methods [Hamilton *et al.*, 2017; Chen *et al.*, 2018; Chen *et al.*, 2017; Chiang *et al.*, 2019; Zeng *et al.*, 2019; Huang *et al.*, 2018] generally solve this issue by running model on the sampled subgraph batches, and offline propagation methods [Wu *et al.*, 2019; Klicpera *et al.*, 2018; Frasca *et al.*, 2020; Zhang *et al.*, 2021] achieve fast training and inference by decoupling feature propagation from prediction as a pre-processing step. Notably, historical embedding methods [Chen *et al.*, 2017; Fey *et al.*, 2021] store intermediate node embeddings from previous training iterations and use them as approximations for accurate embeddings.

3 Preliminaries

In this section we present some model backgrounds and basic notations. Let $G = (\mathcal{V}, \mathcal{E})$ denote a graph, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ is the node set that consists of n vertices and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the edge set. For node $v \in \mathcal{V}$, let $\mathcal{N}(v) = \{v' : v' \in \mathcal{V}, (v, v') \in \mathcal{E}\}$ denote the set of its neighbors. Let each node v_i be associated with a feature vector $\boldsymbol{x}_i \in \mathbb{R}^F$ where F is the hidden dimension, and let $\mathbf{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^\top \in \mathbb{R}^{n \times F}$ denote the feature matrix.

3.1 Transformer

Standard Transformer Layers. Transformer [Vaswani *et al.*, 2017] is first proposed to model sequential text data with consecutive Transformer layers, each of which mainly

consists of a multi-head self-attention (MHA) module and a position-wise feed-forward network (FFN) with residual connections. For queries $\boldsymbol{Q} \in \mathbb{R}^{n_q \times d}$, keys $\boldsymbol{K} \in \mathbb{R}^{n_k \times d}$ and values $\boldsymbol{V} \in \mathbb{R}^{n_k \times d}$, the scaled dot-product attention module can be defined as

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\boldsymbol{A})\boldsymbol{V}, \boldsymbol{A} = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}, \quad (1)$$

where n_q, n_k are number of elements in queries and keys, and d is the hidden dimension. After multi-head attention, the position-wise feed-forward network and layer normalization are performed on the output.

Biased Transformer Layers. A bias term can be added to the attention weights $\boldsymbol{A}$ to represent pair-wise knowledge like relative positional encodings in [Shaw *et al.*, 2018]. Suppose we have a bias matrix $\boldsymbol{B} \in \mathbb{R}^{n_q \times n_k}$, the biased-MHA can be formulated by replacing the standard attention module with the attention weight matrix computed by $\boldsymbol{A} = \frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} + \boldsymbol{B}$.

3.2 Graph Hierarchies

For graph $G^0 = (V^0, E^0)$, graph coarsening aims to find $G^1 = (V^1, E^1)$ that captures the essential substructures of G_0 and is significantly smaller ($|V^1| \ll |V^0|$,$|E^1| \ll |E^0|$). We assume graph coarsening is performed by coalescing nodes with a surjective mapping function $\phi : V^0 \to V^1$. Every node $v_i^1 \in V^1$ corresponds to a node cluster $\phi^{-1}(v_i^1) = \{v_j^0 \in V^0 : \phi(v_j^0) = v_i^1\}$ in G^0, and the edge set of G^1 is defined as $E^1 = \{(v_i^1, v_j^1) : \exists v_r^0 \in \phi^{-1}(v_i^1), v_s^0 \in \phi^{-1}(v_j^1), \text{such that } (v_r^0, v_s^0) \in E^0\}$. We also initialize node embeddings of G^1 by $\boldsymbol{x}_i^1 = \text{Mean}(\{\boldsymbol{x}_j^0 : v_j^0 \in \phi^{-1}(v_i^1)\})$ for every node v_i^1 in V^1. The coarsening ratio α at this step is defined as $\alpha = \frac{|V^1|}{|V^0|}$. By running the coarsening algorithm recursively, a graph hierarchy $\{G^0, G^1, \ldots, G^H\}$ can be constructed to summarize multi-level structures.

4 Proposed Approach

In the following section, we will describe the motivation and approach behind the creation of our HSGT model, and subsequently provide a detailed description of the architecture of the entire model. Figure 2 gives a high-level illustration of model architecture, and Algorithm 1 describes HSGT.

4.1 Motivation for Utilizing Graph Hierarchies

The key difference between HSGT and previous graph Transformers is the utilization of graph hierarchical structure. Previous methods for leveraging high-level context information can only expand the receptive field and increase sample size, which leads to significant computational overhead and performance loss on node-level tasks. In contrast, HSGT utilizes graph hierarchies to communicate high-level information via the embedding of virtual nodes, resulting in several key benefits: (1) low computational cost: the number of high-level virtual nodes is minimal and the updating process is completed via efficient sampling strategies and historical embedding; (2) broad receptive field: the receptive field of a single node includes any node related to its corresponding high-level nodes; (3) high flexibility: one can build the graph hierarchy using any graph coarsening algorithm, and arbitrarily choose the sampling strategy to control the structural information involved in the learning process. In the following paragraphs, we will elaborate on the design of the model and demonstrate its effectiveness through experiments.

Algorithm 1 Overview of HSGT

Input: Input graph $G = (V, E)$, with corresponding hierarchy $\{G^0, G^1, \ldots, G^H\}$, initial feature matrices $\boldsymbol{X}_0, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_H$ and hierarchical mappings $\phi_1, \ldots, \phi_H$, batch size B.
Output: embedding $\boldsymbol{h}_v$ for every node v.

$V_{\text{left}}^H \leftarrow V^H$, where $G^H = (V^H, E^H)$.
while V_{left}^H is not empty **do**
  Sample V_B^H from V_{left}^H with $|V_B^H| = B$.
  $V_{\text{left}}^H \leftarrow V^H \setminus V_B^H$.
  for j in $1, 2, \ldots, H$ **do**
    $V_B^{\tilde{j-1}} \leftarrow \phi_j^{-1}(V_B^j)$.
    $V_B^{j-1} \leftarrow \text{NeighborSample}(V_B^{\tilde{j-1}})$. (Section 4.3)
    $\boldsymbol{H}_{j-1} \leftarrow \boldsymbol{X}_{j-1}[V_B^{j-1}]$.
  end for
  for j in $0, 1, 2, \ldots, H$ **do**
    $\boldsymbol{H}_j \leftarrow \text{HorizontalBlock}(\boldsymbol{H}_j)$.
    if $j < H$ **then**
      $\boldsymbol{H}_{j+1} \leftarrow \text{VerticalBlock}(\boldsymbol{H}_j, \boldsymbol{H}_{j+1})$.
    end if
  end for
  $\boldsymbol{H}_0 \leftarrow \text{ReadoutBlock}(\boldsymbol{H}_0, \boldsymbol{H}_1, \ldots, \boldsymbol{H}_H)$.
  $\boldsymbol{h}_v \leftarrow \boldsymbol{H}_0[v], \forall v \in V_B^H$.
end while
return $\boldsymbol{h}_v, \forall v \in V$.

4.2 Model Architecture

Graph Hierarchy Construction and Input Transformation

For input graph $G^0 = (V^0, E^0)$ with feature matrix $\boldsymbol{X}$, we first use the chosen coarsening algorithm to produce the graph hierarchy $\{G^0, G^1, \ldots, G^H\}$ with initial feature matrices $\boldsymbol{X}_0, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_H$ and corresponding hierarchical mappings $\phi_1, \ldots, \phi_H$. The number of hierarchical layers H and the coarsening ratios for each step $\alpha_1, \ldots, \alpha_H$ are predefined as hyperparameters. The model imposes no limitations on the specific graph coarsening algorithm. In our implementation, we choose METIS [Karypis and Kumar, 1998] which is designed to partition a graph into mutually exclusive groups and minimize the frequency of inter-links between different groups. The chosen modern METIS algorithm is fast and highly scalable with time complexity approximately bounded by $O(|E|)$, and only needs to compute once at the pre-processing stage. In experiments, even partitioning of the largest *ogbn-products* graph is finished within 5 minutes, bringing almost no overhead to the entire training process.

We also apply linear transformations and degree encodings for initial features before they are fed into Transformer

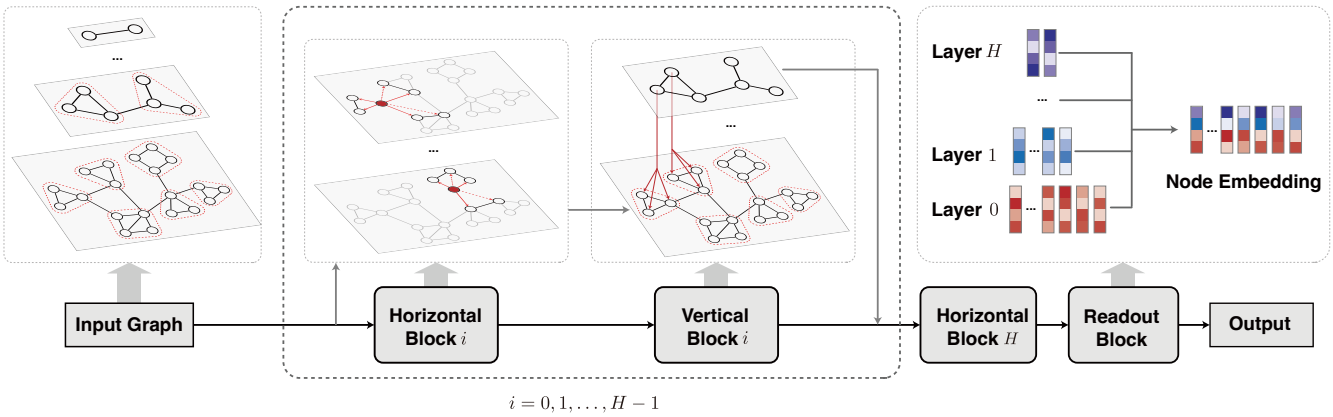

Figure 2: A high-level illustration of the proposed HSGT model architecture. At every hierarchical layer, a horizontal block first exchanges and transforms information in each node's local context, then a vertical block is performed to adaptively coalesce every substructure if a higher layer exists. Finally, a readout block aggregates multi-level representations to calculate the final output.

blocks. All feature vectors in $\boldsymbol{X}_0, \boldsymbol{X}_1, \ldots, \boldsymbol{X}_H$ are projected into $\mathbb{R}^d$ using a linear layer, where d is the hidden size. Additionally, inspired by centrality encoding in [Ying *et al.*, 2021], we add learnable embedding vectors indexed by node degree to the transformed node feature at layer 0 to represent structural information.

Horizontal Block

In every horizontal block, we aim to horizontally aggregate and transform node representations in every node's local context using structural-enhanced Transformer layers. Suppose the input graph is $G = (V, E)$ with n nodes, the feature matrix is $\boldsymbol{H}$, and every node v has an individual local receptive field $\mathcal{R}(v) \subset V$ which we will discuss later. Following [Ying *et al.*, 2021], to leverage graph structure into self-attention, we choose to quantify the connectivity between nodes with the distance of the shortest path (SPD), and use an attention bias term to represent the structural information. To reduce the computational cost, we set a maximum SPD length D such that SPDs longer than D will not be computed. We also mask nodes not in $\mathcal{R}(v)$ out of the receptive field of v by setting the corresponding bias term to $-\infty$. Formally, the bias matrix $\boldsymbol{B} \in \mathbb{R}^{n \times n}$ is defined as

$$\boldsymbol{B}_{i,j} = \begin{cases} b_{\mathrm{SPD}(v_i, v_j)}, & \text{if } v_j \in \mathcal{R}(v_i), \mathrm{SPD}(v_i, v_j) \le D, \\ -\infty, & \text{if } v_j \notin \mathcal{R}(v_i), \\ 0, & \text{else.} \end{cases}$$

where $\boldsymbol{B}_{i,j}$ is the (i, j)-element of $\boldsymbol{B}$, $b_0, b_1, \ldots, b_D \in \mathbb{R}$ are a series of learnable scalars. Then the horizontal block is built by stacking the following Transfomer layers:

$$\boldsymbol{H} = \text{Biased-Transformer}(\boldsymbol{H}, \boldsymbol{H}, \boldsymbol{H}, \boldsymbol{B}). \tag{2}$$

Compared with GNNs, our method promotes a high-order knowledge aggregation in a broader context while leveraging the complete structural information. In the case of full-batch training on small graphs, every node can have a global receptive field, i.e. $\mathcal{R}(v) = V, \forall v \in V$. But during sampling-based training on huge graphs, the receptive field of each node is restricted to the current batch $V_B \subset V$, and we empirically discover that due to the unbalance and irregularity of sampling methods, computing every pair-wise attention in V_B will lead to significant performance drop on node-level tasks. In the mean time, intra-batch communication will be limited if we set the receptive field of each node to its local neighbors. To balance the two aspects, we choose to form the receptive field $\mathcal{R}(v)$ of node v with its D-hop neighbors $\mathcal{N}_D(v)$ and nodes individually randomly sampled from V_B by probability p. Besides, during experiments we discover that by sharing parameters among horizontal blocks at different hierarchical layers, we can greatly reduce the number of model parameters while the model performance is not affected. This is probably due to the structural similarity between graphs at different hierarchical levels and the strong expressive capacity of Transformer layers. We will test the effectiveness of the two strategies in ablation studies.

Vertical Block

The vertical block focuses on aggregating node representations produced by the previous horizontal block and generating embeddings for nodes at next hierarchical level. In contrast to simple pooling functions like mean and sum, to overcome their incapability of capturing important nodes and substructures, we reuse the attention mechanism to adaptively merge vector embeddings. Suppose the vertical block aims to calculate embeddings for nodes in G^{i+1}. For node $v \in V^{i+1}$ with transformed initial feature $\boldsymbol{x}_v$, its representation $\boldsymbol{h}_v$ after the vertical aggregation is computed as

$$\boldsymbol{N}_v = \text{Stack}(\{\boldsymbol{h}_s : s \in \phi_{i+1}^{-1}(v)\}), \tag{3}$$

$$\boldsymbol{h}_v = \text{Transformer}(\boldsymbol{x}_v, \boldsymbol{N}_v, \boldsymbol{N}_v), \tag{4}$$

where $\boldsymbol{x}_v, \boldsymbol{h}_v$ are viewed as matrices in $\mathbb{R}^{1 \times d}$, and the vertical block computes representation for every node in the input. This aggregation scheme allows every fused representation $\boldsymbol{h}_v$ to contain meaningful information on its corresponding low-level substructure, helping the next horizontal block to achieve better high-level knowledge exchange.

Readout Block

After the horizontal blocks are performed on every hierarchical level, we use the readout block to fuse representations

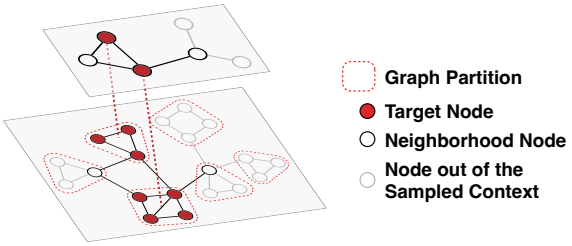

Figure 3: An illustration of the proposed sampling method.

and produce final node embeddings. This operation brings well aggregated multi-level information to nodes in the original graph, expanding their receptive field to a much higher scale. Formally, let $\boldsymbol{h}_r$ denote the embedding of node r at level l generated by the l-th horizontal block, then for a node $v \in V^0$, its final output embedding $\overline{\boldsymbol{h}}_v$ is calculated through attention with its corresponding high-level nodes:

$$\boldsymbol{Z}_v = [\boldsymbol{h}_{t_0}, \boldsymbol{h}_{t_1}, \ldots, \boldsymbol{h}_{t_H}]^\top, \tag{5}$$

$$\overline{\boldsymbol{h}}_v = \text{Transformer}(\boldsymbol{h}_v, \boldsymbol{Z}_v, \boldsymbol{Z}_v), \tag{6}$$

where $t_0 = v, t_j = \phi_j(t_{j-1})$, for $j = 1, \ldots, H$.

4.3 Training Strategy

Hierarchical Sampling

When training on large-scale graphs, at every step, our model can only be operated on a sampled batch $\{G_B^0, G_B^1, \ldots, G_B^H\}$ of the entire graph hierarchy $\{G^0, G^1, \ldots, G^H\}$, that G_B^i is a subgraph of G^i for every $i = 0, 1, \ldots, H$. To keep every substructure in low-level hierarchical layers complete, we follow a top-to-bottom sampling approach. For batch size b, we first randomly sample b nodes at V^H as $\tilde{V}_B^H$, then recursively sample nodes from layer $H-1$ to 0 in $\tilde{V}_B^{i-1} = \bigcup_{v \in \tilde{V}_B^i} \phi_i^{-1}(v), i = H, H-1, \ldots, 1$. Here we call nodes in $\{\tilde{V}_B^0, \tilde{V}_B^1, \ldots, \tilde{V}_B^H\}$ *target nodes*, because only representations of nodes in $\tilde{V}_B^0$ will be used for supervised learning and $\{\tilde{V}_B^1, \ldots, \tilde{V}_B^H\}$ contains all high-level nodes they relate to. Meanwhile, to promote local context interaction in horizontal blocks, we additionally sample a neighborhood set for every target node at all levels using neighbor sampling. An illustration of this process is presented in Figure 3. We construct the final sampled nodes set $\{V_B^0, V_B^1, \ldots, V_B^H\}$ by adding the neighborhood sets to the target nodes set, and $\{G_B^0, G_B^1, \ldots, G_B^H\}$ are the corresponding induced subgraphs. The resulting sampling strategy allows the model to operate on complete hierarchical structures with local context preserved, which is critical for the horizontal and vertical modules to work well.

Historical Embeddings for High-level Nodes

The sampling scheme above could cause issues. For example, for neighborhood node $v \in V_B^1 \setminus \tilde{V}_B^1$, it is very likely that most of its corresponding nodes at layer 0 will not appear in the sampled nodes set V_B^0, since we do not deliberately add $\phi_1^{-1}(v)$ to the sampled set as the target nodes do. Thus, it is not possible to directly get the representation of v by aggregating embeddings of nodes in $\phi_1^{-1}(v)$ via vertical blocks when some of $\phi_1^{-1}(v)$ do not exist in the sampled context. And manually adding those nodes to the data batch will lead to a massive expansion of the computational graph (almost $10\times$). Nevertheless, if we skip the neighborhood sampling step for high-level nodes, the inter-batch communication of structural knowledge could be baffled, which contradicts with our initial goal.

To disentangle such multi-level dependencies, we utilize the historical embedding method proposed in [Chen *et al.*, 2017; Fey *et al.*, 2021] to alleviate the *neighbor explosion* in GNNs. In our model, the historical embeddings act as an offline storage $\mathcal{S}$ of high-level nodes (above level 0), which is accessed and updated at every batch using *push* and *pull* operations. At every batch, the vertical blocks are only performed on the target nodes, and we push the newly aggregated embeddings for high-level target nodes to $\mathcal{S}$. For horizontal blocks on high-level nodes, we approximate the embeddings of neighborhood nodes via pulling historical embeddings in $\mathcal{S}$ acquired in previous batches. Specifically, for horizontal block i, its input $\boldsymbol{H}$ will be computed as

$$\boldsymbol{H} = \text{Stack}(\{\boldsymbol{h}_s : s \in V_B^i\}), \tag{7}$$

$$\approx \text{Stack}(\{\boldsymbol{h}_s : s \in \tilde{V}_B^i\} \cup \{\tilde{\boldsymbol{h}}_s : s \in V_B^i \setminus \tilde{V}_B^i\}), \tag{8}$$

where $\boldsymbol{h}_s$ denotes accurate embedding of s calculated by the previous horizontal block, and $\tilde{\boldsymbol{h}}_s$ denotes the historical embedding of s from previous batches. With historical embeddings, we enable the inter-batch communication of high-level contexts with low extra computational cost and high accuracy bounded by theoretical results in [Chen *et al.*, 2017; Fey *et al.*, 2021]. Our approach is the first implementation of the historical embedding method on graph Transformer models, and its effectiveness is further demonstrated by the ablation studies.

5 Experiments

In this section we first evaluate HSGT on different benchmark tasks, and then perform ablation studies, scalability tests, and parameter analysis.

5.1 Node Classification Tasks

Datasets. We conduct experiments on nine benchmark datasets including four small-scale datasets (Cora, CiteSeer, PubMed [Sen *et al.*, 2008; Yang *et al.*, 2016], Amazon-Photo [Shchur *et al.*, 2018]) and six large-scale datasets (ogbn-arxiv, ogbn-proteins, ogbn-products [Hu *et al.*, 2020], Reddit [Hamilton *et al.*, 2017], Flickr, Yelp [Zeng *et al.*, 2019]). We use the predefined dataset split if possible, or we set a random 1:1:8 train/valid/test split.

Baselines and Settings. We compare HSGT against a wide range of baseline scalable graph learning methods including GCN [Kipf and Welling, 2016], GAT [Veličković *et al.*, 2017], GIN [Xu *et al.*, 2018], GraphSAGE [Hamilton *et al.*, 2017], Cluster-GCN [Chiang *et al.*, 2019], GraphSAINT [Zeng *et al.*, 2019], GAS-GCN [Fey *et al.*, 2021], SIGN [Frasca *et al.*, 2020], GraphZoom [Deng *et al.*, 2019]

#nodes	2.7K	3.3K	19.7K	7.6K	169K	233K	89.2K	716K	133K	2.4M
#edges	5.2K	4.5K	44.3K	119K	1.1M	11.6M	450K	7.0M	40M	61.8M
Dataset	CORA	CITESEER	PUBMED	AMAZON-PHOTO	ogbn-arxiv	REDDIT	FLICKR	YELP	ogbn-proteins	ogbn-products
GCN	77.20±1.51	69.49±0.58	77.60±0.96	92.44±0.22	71.74±0.29*	91.01±0.29	51.86±0.10	32.14±0.66	72.51±0.35*	75.64±0.21*
GAT	82.80±0.47	69.20±0.45	76.90±0.85	92.88±0.37	57.88±0.18	96.50±0.14	52.39±0.05	61.58±1.37	72.02±0.44*	79.45±0.59
GIN	75.93±0.99	63.83±0.49	77.03±0.42	80.14±1.46	52.23±0.34	86.37±0.62	48.28±0.85	29.75±0.86	70.76±0.08	74.79±0.81
GraphSAGE	-	-	-	-	71.49±0.27*	96.53±0.11	51.86±0.35	53.89±0.85	77.68±0.20*	78.50±0.14*
Cluster-GCN	-	-	-	-	69.76±0.49*	95.12±0.08	50.25±0.83	52.50±0.19	74.89±0.12	78.97±0.33*
GraphSAINT	-	-	-	-	58.63±0.33	90.92±0.61	51.91±0.06	56.22±1.14	70.22±0.84	79.08±0.24*
GAS-GCN	82.29±0.76*	**71.18**±0.97*	79.23±0.62*	90.53±1.40*	71.68*	95.45*	54.00*	62.94*	-	76.66*
SIGN	-	-	-	-	-	96.8±0.0*	51.4±0.1*	63.1±0.3*	-	77.60±0.13*
GraphZoom	-	-	-	-	71.18±0.18*	92.5*	-	-	-	74.06±0.26*
Graphormer	66.35±2.44	56.22±3.27	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
Graphormer-SAMPLE	75.14±1.31	61.46±1.90	75.45±0.98	92.76±0.59	70.43±0.20	93.05±0.22	51.93±0.21	60.01±0.45	72.34±0.51	79.10±0.12
SAN	36.61±3.49	44.35±1.08	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
SAT	72.40±0.31	60.93±1.25	OOM	OOM	OOM	OOM	OOM	OOM	OOM	OOM
SAT-SAMPLE	74.55±1.24	61.58±0.87	76.70±0.74	91.35±0.42	68.20±0.46	93.37±0.32	50.48±0.34	60.32±0.65	70.62±0.85	77.64±0.20
ANS-GT	79.35±0.90	64.52±0.71	77.80±0.65	80.41±0.78	72.34±0.50	95.30±0.81	-	-	74.67±0.65	80.64±0.29
HSGT	**83.56**±1.77	67.41±0.92	**79.65**±0.52	**95.01**±0.34	**72.58**±0.31	**97.30**±0.24	**54.12**±0.51	**63.47**±0.45	**78.13**±0.25	**81.15**±0.13

Table 1: Results on node classification datasets. OOM stands for out of memory. * indicates results cited from the original papers and the OGB leaderboard.

and graph Transformers including Graphormer [Ying *et al.*, 2021], SAN [Kreuzer *et al.*, 2021], SAT [Chen *et al.*, 2022] and ANS-GT [Zhang *et al.*, 2022]. For GCN, GAT and GIN, we perform full-batch training on small-scale datasets and sampling-based training on large-scale datasets. By default, Graphormer, SAN and SAT require full-batch training, which is prohibited by GPU memory bound in most cases. We also add the Graphormer-SAMPLE and SAT-SAMPLE baselines that perform the model on subgraphs generated from neighborhood sampling, as mentioned in the introduction. For all experiments, the overhead of preprocessing steps (including METIS partition) does not exceed 5 minutes. The detailed settings for baselines and HSGT are listed in the appendix. We report the means and standard deviations of performances on test set initialized by three different random seeds.

Results. We present the node classification performances in Table 1, where the metric for OGBN-PROTEINS is roc-auc while the metric for other datasets is acc. On small-scale datasets where graph Transformer baselines are mostly outperformed by GNN methods, HSGT delivers competitive performance against the GNN baselines. While on large-scale datasets, HSGT performs consistently better than all GNN baselines, achieving state-of-the-art and showing that the Transformer architecture is well capable of handling large-scale node-level tasks. Overall, the results have also demonstrated the outstanding generalizability of the HSGT method on multi-scale graph data.

It can also be observed that Transformers generally perform bad at node-level tasks on small graphs probably because the global attention introduces much irrelevant noise. And the Graphormer-SAMPLE and SAT-SAMPLE baseline fail to produce satisfactory results on multi-level benchmarks since a naive sampling-based approach can not capture the necessary high-level contextual information. On the contrary, on all datasets HSGT performs significantly better than the Graphormer-SAMPLE and SAT-SAMPLE baseline, showing the effectiveness of our proposed graph hierarchical structure and training strategies. Notably, the performance gains of HSGT are greater on large-scale datasets than small-scale ones, indicating that a large amount of data is crucial in optimizing the performance potentials of Transformer.

5.2 Ablation Studies

Settings. We design four HSGT variants to demonstrate the benefits of vertical blocks, structural encodings, historical embeddings and readout blocks, respectively, while other model settings stay unchanged. In Table 2, *w/o vertical blocks*: the vertical feature aggregation is performed with the simple mean function, instead of a Transformer block. *w/o structural encodings*: all SPD attention biases are removed. *w/o historical embeddings*: neighborhood nodes of high-level nodes are no longer sampled, then historical embeddings are no longer needed. *w/o readout blocks*: the multi-level readout is performed by concatenating feature vectors at different levels. *w/o parameter sharing*: all horizontal blocks and vertical blocks have an individual set of parameters. *random partition*: the coarsening process is performed via random partition, instead of METIS algorithm.

	FLICKR	YELP	ogbn-products
w/o vertical blocks	52.85	62.46	78.01
w/o structural encodings	49.11	59.78	77.05
w/o historical embeddings	52.04	61.79	80.72
w/o readout blocks	52.58	62.01	80.34
w/o parameter sharing	53.01	63.44	81.13
random partition	43.21	60.77	75.23
HSGT	**53.02**	**63.47**	**81.15**

Table 2: Results of ablation studies.

Results. Table 2 summarizes the results of ablation studies. Most variants suffer from performance loss, showing that all tested modules are necessary to raise the performance of HSGT to its best level. If we replace Transformer layer in vertical and readout blocks with simple operations like mean, then multi-scale information can not be adaptively fused. And the model will not be able to recognize graph structure when we remove the necessary structural encodings, which explains the severe performance drop we witness. It can also be observed that if we take out the neighborhood sampling of high-level nodes to avoid historical embeddings, the high-level context information exchange can be blocked, resulting in performance drop on large-scale datasets. When individual learnable parameters are assigned to horizontal blocks at dif-

Model	Settings	Peak GPU Memory Usage	#Parameters	Inference Time	Performance
GraphSAGE	$l = 2, d = 64$	1064MB	16.4K	48.6s	75.60
	$l = 2, d = 128$	1150MB	65.5K	53.7s	77.38
	$l = 2, d = 256$	1316MB	262.1K	70.2s	77.58
	$l = 3, d = 128$	1928MB	98.3K	103.3s	78.25
Graphormer-SAMPLE	$l = 2, d = 128$	1874MB	223.6K	90.6s	63.44
	$l = 2, d = 256$	2032MB	840.2K	93.5s	64.76
HSGT	$l = 2, d = 64$	1903MB	112.7K	97.4s	80.99
	$l = 2, d = 128$	2208MB	421.1K	99.8s	81.10
	$l = 3, d = 64$	3374MB	137.7K	108.2s	**81.15**

Table 3: Results of scalability tests on `ogbn-products` dataset.

ferent levels, the model performance is almost not affected while the number of parameters could increase by at least $1.5\times$. A random coarsening approach also causes the model performance to drop dramatically because HSGT requires high-quality structural partitions calculated by the METIS algorithm to capture and aggregate high-level information.

5.3 HSGT Efficiently Scales to Large Graphs

Settings. To comprehensively examine HSGT's scalability to large-scale graphs, we perform tests on the largest `ogbn-products` dataset that contains over 2.4 million nodes against the standard GraphSAGE method and Graphormer-SAMPLE above. To give a fair comparison, for GraphSAGE and Graphormer-SAMPLE we set the batch size to 400, and for HSGT we keep layer 0 nodes per batch to around 400. In Table 3, l stands for the number of layers for GraphSAGE and Graphormer-SAMPLE, while number of Transformer layers at horizontal blocks for HSGT. d stands for the hidden dimension for all models. Other model parameters stay the same with experiments in Table 1. We use built-in PyTorch CUDA tools to measure peak GPU memory usage during experiments. For the number of parameters, we calculate the number of all learnable model parameters except input and output projections.

Results. In Table 3 we list the results. Traditionally, it is believed that the Transformer architecture tends to achieve high performance with high computational complexity and lots of parameters. However, experimental results show that the proposed HSGT model can achieve outstanding performance with reasonable costs that are similar to the widely-used GraphSAGE method, showing that HSGT can be efficiently scaled to large graphs with normal computational resources. Even under the lightest setting $l = 2, d = 64$, HSGT is capable of delivering results higher than all GNN baselines while keeping moderate GPU memory usage and parameter size, which can be attributed to the small hidden size (64) and the parameter sharing among horizontal blocks. The Graphormer-SAMPLE may cost fewer resources at lightweight configurations since HSGT has the additional horizontal and vertical blocks, but it is significantly outperformed by both GraphSAGE and HSGT.

5.4 Parameter Analysis

Coarsening Ratios

The coarsening ratios $\alpha_1, \ldots, \alpha_H$ are used as parameters for the METIS algorithm to generate initial graph hierarchy $\{G^0, G^1, \ldots, G^H\}$. Normally we set the number of additional hierarchical layers H to 1 or 2, and coarsening ratios are picked from $\{0.2, 0.1, 0.05, 0.02, 0.01, 0.005\}$. Other settings stay the same with models in Table 1. Here we perform experiments on dataset Flickr, Yelp and ogbn-products to study the influence of coarsening ratios, and we list the results in Table 4. It can be observed that the best setting for coarsening ratios and batch size could vary for different datasets.

Coarsening Ratios	FLICKR	YELP	`ogbn-products`
$\{0.005\}$	**52.71**	61.98	**81.15**
$\{0.002\}$	51.32	61.44	80.55
$\{0.1, 0.1\}$	47.59	62.95	OOM
$\{0.1, 0.2\}$	47.03	**63.47**	OOM

Table 4: Results of coarsening ratio tests.

Intra-batch Connectivity

At previous sections we have mentioned that in horizontal blocks, we construct the receptive field of each node with its D-hop neighbors and nodes randomly sampled from the sampled batch by probability p. Here we study the impact of p value with experiments and summarize the results in Table 5, where other settings stay the same with those in Table 1. From the results we can see that as p varies from 0 to 1, the model performance generally increases until it reaches a peak and then decreases, which corresponds to our previous analysis.

Intra-batch Connectivity	FLICKR	YELP	`ogbn-products`
$p = 0.0$	51.04	62.83	80.45
$p = 0.1$	50.80	**63.47**	**81.15**
$p = 0.3$	**53.02**	62.92	80.67
$p = 0.5$	48.47	60.41	80.14
$p = 1.0$	45.83	60.74	78.65

Table 5: Results of intra-batch connectivity tests.

6 Conclusion

In this paper we propose HSGT, a Transformer-based neural architecture for scalable graph learning, and our model has shown strong performance and generalizability on multi-scale graph benchmarks with reasonable computational costs.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grant No. 62276006).

References

[Chen *et al.*, 2017] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.

[Chen *et al.*, 2018] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

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

[Chiang *et al.*, 2019] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 257–266, 2019.

[Deng *et al.*, 2019] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. *arXiv preprint arXiv:1910.02370*, 2019.

[Dwivedi and Bresson, 2020] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[Fey *et al.*, 2021] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. *arXiv preprint arXiv:2106.05609*, 2021.

[Frasca *et al.*, 2020] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[Hu *et al.*, 2020] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[Huang *et al.*, 2018] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *arXiv preprint arXiv:1809.05343*, 2018.

[Karypis and Kumar, 1998] George Karypis and Vipin Kumar. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN*, 38:7–1, 1998.

[Kipf and Welling, 2016] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[Klicpera *et al.*, 2018] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.

[Kreuzer *et al.*, 2021] Devin Kreuzer, Dominique Beaini, William L Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *arXiv preprint arXiv:2106.03893*, 2021.

[Rampášek *et al.*, 2022] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *arXiv preprint arXiv:2205.12454*, 2022.

[Rong *et al.*, 2020] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. Self-supervised graph transformer on large-scale molecular data. *arXiv preprint arXiv:2007.02835*, 2020.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[Shaw *et al.*, 2018] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[Shchur *et al.*, 2018] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[Veličković *et al.*, 2017] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[Wu *et al.*, 2019] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[Wu *et al.*, 2022] Qitian Wu, Wentao Zhao, Zenan Li, David Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. In *Advances in Neural Information Processing Systems*, 2022.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[Yang *et al.*, 2016] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[Ying *et al.*, 2021] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform bad for graph representation? *arXiv preprint arXiv:2106.05234*, 2021.

[Zeng *et al.*, 2019] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[Zhang *et al.*, 2021] Wentao Zhang, Ziqi Yin, Zeang Sheng, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. Graph attention multi-layer perceptron. *arXiv preprint arXiv:2108.10097*, 2021.

[Zhang *et al.*, 2022] Zaixi Zhang, Qi Liu, Qingyong Hu, and Chee-Kong Lee. Hierarchical graph transformer with adaptive node sampling. *arXiv preprint arXiv:2210.03930*, 2022.

[Zhao *et al.*, 2021] Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification. *arXiv preprint arXiv:2110.13094*, 2021.