HubGT: Scalable Graph Transformer with Decoupled Hierarchy Labeling

[Scalable Data Science]

Ningyi Liao*
Nanyang Technological University
Singapore
liao0090@e.ntu.edu.sg

Zihao Yu*
Nanyang Technological University
Singapore
zihao010@e.ntu.edu.sg

Siqiang Luo
Nanyang Technological University
Singapore
siqiang.luo@ntu.edu.sg

Gao Cong
Nanyang Technological University
Singapore
gaocong@ntu.edu.sg

ABSTRACT

Graph Transformer (GT) has recently emerged as a promising neural network architecture for learning graph-structured data. However, its global attention mechanism with quadratic complexity concerning the graph scale prevents wider application to large graphs. Effectively representing graph information while ensuring learning efficiency remains challenging, as our analysis reveals that current GT designs targeting scalability still suffer from the computational bottleneck related to graph-scale operations. In this work, we tackle the GT scalability issue by proposing HubGT, a scalable Graph Transformer boosted by fully decoupled graph processing and simplified learning. HubGT represents the graph by a novel hierarchical scheme exploiting hub labels, which is shown to be more informative than plain adjacency by offering global connections while promoting locality, and is particularly suitable for handling complex graph patterns such as heterophily. We also design algorithms for efficiently constructing and querying the hub label hierarchy tailored for the GT attention training in scalable deployments. Notably, the precomputation and training processes of HubGT achieve complexities *linear* to the number of graph edges and nodes, respectively, while the training stage completely removes graph-related computations, leading to favorable mini-batch capability and GPU utilization. Extensive experiments demonstrate that HubGT is efficient in terms of computational enhancement and mini-batch capability over existing GT designs on large-scale benchmarks, while achieving top-tier effectiveness on both homophilous and heterophilous graphs.

PVLDB Reference Format:
Ningyi Liao*, Zihao Yu*, Siqiang Luo, and Gao Cong. HubGT: Scalable Graph Transformer with Decoupled Hierarchy Labeling. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/nyLiao/HubGT.

*Both authors contributed equally to this research.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Graph Transformers (GTs) characterize a family of neural networks that introduce the advantageous Transformer architecture [42] to the realm of graph data learning. These models have garnered increasing research interest for tasks such as knowledge graph retrieval, molecule analysis, and Large Language Model alignment [21, 49, 51, 55]. Despite their achievements, vanilla GTs are highly limited to specific tasks because of the full-graph attention mechanism, which has computational complexity at least quadratic to the graph size, rendering it impractical for a single graph with more than thousands of nodes. Enhancing the scalability of GTs is thus a prominent task for enabling these powerful models to handle a wider range of graph data at large scales.

To scale up Graph Transformers, existing studies explore various strategies to divide and represent the graph structure into smaller batches and employ mini-batch training. One approach is to simplify the Transformer *architecture* with a specialized attention module based on graph topology [38, 46, 47], which learns on existing edge connections instead of all-pair interactions. Alternatively, sophisticated *representations* are developed for inputting graph information to GT models as node embeddings and positional encodings. These works feature graph processing techniques such as adjacency-based spatial propagation [10, 25], polynomial spectral transformation [15, 33], and hierarchical graph coarsening [52, 54]. However, we identify two major drawbacks of existing scalable GTs: in terms of efficacy, most models primarily concentrate on local adjacency, which undermines GT expressivity in capturing full-graph information; in terms of efficiency, graph-scale operations still persist throughout their training iterations, and the overhead substantially increases as the graphs become larger.

In this work, we propose HubGT, a scalable Graph Transformer exploiting the hub labeling technique to produce rich hierarchical graph information with efficient computation. HubGT is inspired by the well-studied concept of graph labeling, which identifies important hubs in the graph structure and imparts fast computation of the shortest path distance (SPD) for node pairs [4, 28, 50]. We innovatively introduce the graph label hierarchy to enhance GT capability, which is superior to the conventional adjacency-based scheme in establishing global connections to influential graph hubs. To further represent node-pair interactions, HubGT is the pioneering work to employ SPD as positional encoding for large-scale GTs, which is only practicable under its efficient calculation. The expressive representations empower HubGT to capture graph knowledge beyond edges and excel in complex graph data patterns ranging from homophily to heterophily.

To efficiently construct graph labels and calculate SPD for GT learning, we design a three-level hierarchical index orienting the specific query requirement in HubGT, which leverages both local connections and global hubs to facilitate distance computation. Graph indexing can be fully decoupled as precomputation and constructed with $O(m)$ overhead. Then, querying SPD on the index can be performed in $O(1)$ time, rendering HubGT learning as simple as training normal Transformers under $O(n)$ complexity without interweaving graph topology, where m and n are the numbers of graph edges and nodes, respectively. Both precomputation and training of HubGT achieve theoretical complexities on par with the respective state-of-the-art GTs and are significantly faster in practice thanks to the simplicity of hub labeling.

We summarize the contributions of this work as follows:

- We propose HubGT as a scalable Graph Transformer with novel hierarchical sampling based on the hub labels, effectively embedding local and global graph topology. We also enable the powerful SPD positional encoding on large-scale graphs.
- We design a hierarchical index dedicated to HubGT graph processing, featuring construction under linear complexity and $O(1)$ query overhead. The decoupled precomputation and simple training contribute to scalable mini-batch GT training.
- We conduct comprehensive experiments to evaluate the effectiveness and efficiency of HubGT on up to million-scale graphs. HubGT achieves top-tier accuracy and demonstrates competitive scalability, especially demonstrating the fastest inference speed.

2 PRELIMINARIES AND RELATED WORKS

Scalable Graph Transformer. Consider a connected graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. The node attribute matrix is $\boldsymbol{X} \in \mathbb{R}^{n \times F_0}$, where F_0 is the dimension of input attributes. A Transformer layer [42] first projects three representations given an input matrix $\boldsymbol{H} \in \mathbb{R}^{n \times F}$:

$$\boldsymbol{Q} = \boldsymbol{H}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{H}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{H}\boldsymbol{W}_V, \tag{1}$$

where $\boldsymbol{W}_Q \in \mathbb{R}^{F \times F_K}$, $\boldsymbol{W}_K \in \mathbb{R}^{F \times F_K}$, $\boldsymbol{W}_V \in \mathbb{R}^{F \times F_V}$ are learnable weights. For a multi-head self-attention module with N_H heads, each attention head possesses its own representations $\boldsymbol{Q}_i, \boldsymbol{K}_i, \boldsymbol{V}_i, i = 1, \cdots, N_H$, and then the output $\tilde{\boldsymbol{H}}$ across all heads is calculated as:

$$\tilde{\boldsymbol{H}}_i = \mathrm{softmax}\left(\frac{\boldsymbol{Q}_i \boldsymbol{K}_i^\top}{\sqrt{F_K}} + \boldsymbol{P}\right)\boldsymbol{V}_i, \quad \tilde{\boldsymbol{H}} = (\tilde{\boldsymbol{H}}_1 \| \cdots \| \tilde{\boldsymbol{H}}_{N_H})\boldsymbol{W}_O, \tag{2}$$

where $\cdot\|\cdot$ denotes the matrix concatenation operation, and $\boldsymbol{P}$ is the optional positional encoding introduced in the next subsection. The projection dimension is usually set as $F_K = F_V = F/N_H$. The majority of GTs [17, 49, 51] are proposed for graph-level learning tasks on small graphs with full-batch training (FB). This is because of the critical scalability bottleneck indicated by Eq. (2), that representing n nodes leads to $O(n^2F)$ time and memory overhead.

To mitigate the quadratic complexity and foster scalable learning, scalable GTs employ mini-batch training, which replaces the full graph representation $\boldsymbol{H}$ with batches containing n_b nodes in Eqs. (1) and (2) and reduces memory complexity to $O(n_b^2 F)$. *Kernal-based GTs* [15, 38, 46, 47] generate batches by neighborhood sampling (NS) and utilize graph kernels, i.e., functions modeling node-pair relations, for attention computation to exploit edge connections. Typically, they necessitate iterative processing of graph data with $O(LmF)$ complexity throughout learning. When the graph scale is large, this term becomes dominant since the edge size m is significantly larger than the node size n. Hence, we argue that such a design is not sufficiently scalable. Another approach to harness mini-batching is through random sampling (RS), which does not necessitate graph information and has no additional overhead. It requires the topology information to be embedded with other designs in an permutation-invariant manner, as introduced in the following subsection. The model can enjoy better scalability if the graph processing is fully independent of GT attention.

Table 1: Complexity analysis on GT models. Training complexity indicates the forward computation per epoch on the full graph. Data scale is represented by the largest node size n and edge size m used in the original papers.

Taxonomy	Model	Train Time	Scale
Vanilla (FB)	Graphormer [51]	$O(Ln^2F)$	0.3K/0.6K
	GRPE [37]	$O(Ln^2F)$	0.3K/0.6K
Kernel-based (NS)	GraphGPS [38]	$O(LnF^2 + LmF)$	1.0K/3.0K
	NodeFormer [47]	$O(LnF^2 + LmF)$	2.4M/60M
	DIFFormer [46]	$O(LnF^2 + LmF)$	1.6M/40M
	PolyNormer [15]	$O(LnF^2 + LmF)$	2.4M/0.1B
Hierarchical (RS)	NAGphormer [10]	$O(LnF^2)$	2.4M/60M
	PolyFormer [33]	$O(LnF^2)$	0.2M/30M
	ANS-GT [52]	$O(LnF^2 + Lns^2F + Lm)$	20K/0.2M
	GOAT [25]	$O(LnF^2 + LmF)$	2.9M/0.1B
	HSGT [54]	$O(LnF^2 + LmF)$	2.4M/0.1B
	HubGT (ours)	$O(LnF^2 + Lns^2F)$	1.6M/0.1B

Graph Embedding and Encoding. Graph topology can be incorporated into the GT attention module through *positional encoding*, which adds an bias term $\boldsymbol{P}$ representing pair-wise information in the attention calculation Eq. (2). Typical encoding approaches include graph proximity [8, 51, 52], Laplacian eigenvectors [17, 21, 26], and shortest path distance [9, 37, 53]. The complexity of such design depends on the specific acquisition of the pair-wise encoding.

Alternatively, *hierarchical GTs* exploit the power of GTs to learn node relations by embedding node-level identity in representation $\boldsymbol{H}$ through the input data $\boldsymbol{X}$. Its core design is crafting an effective embedding scheme to comply with GT expressivity. To realize this, NAGphormer [10], PolyFormer [33], and GOAT [25] look into features using adjacency propagation, spectral graph transformation, and feature space projection, respectively. ANS-GT [52] adaptively samples s-node subgraphs, while HSGT [54] leverages coarsening algorithms. Ideally, hierarchical GTs can process graph topology in $O(m)$ complexity in precomputation and employ RS during training for better scalability. Nonetheless, we note that existing models, except for NAGphormer and PolyFormer, still involve graph-level operations during training as analyzed in Table 1, which hinders GPU utilization and causes additional overhead.

Scalability and Heterophily Issues in GNNs. The scalability issue has been extensively examined for convolutional Graph Neural Networks (GNNs) [23, 43], featuring the decoupled processing on some of the largest graph datasets with linear or even sub-linear complexity [12, 24, 30, 45]. Graph simplification techniques including sampling [11, 13, 18, 56] and coarsening [7, 16, 20] are also explored for reducing the graph scale at different hierarchy levels.

Heterophily is another prominent drawback of common GNNs, which describes the scenarios that neighboring nodes belong to different classes, and the inductive bias in these models are no longer effective. Addressing the issue usually requires tailored management of the underlying graph hierarchy beyond edges, typically realized by enhancing convolution operations [27, 29, 44] or augmenting the graph topology [6, 34]. A recent work [22] also employs the idea of hub labeling to refine graph convolution. Although the high-level idea of designing convolutional GNNs is helpful for GTs, Transformer-based models are unique in respect to their graph data utilization and architectural bottlenecks, and hence require specific techniques for addressing these issues.

3 EFFICIENT HIERARCHICAL LABELING

In this section, we introduce the three-level HubGT hierarchy based on hub labeling and its indexing and querying algorithms. Utilization of the hierarchy for graph embedding and encoding during GT learning will be elaborated in the next section.

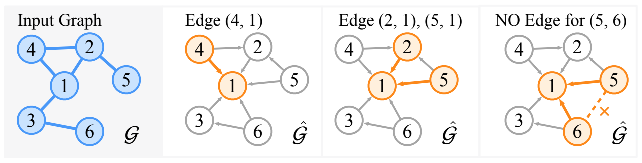
3.1 Problem Statement

Motivation: Graph hierarchy beyond adjacency. The expressiveness of Graph Transformers mainly stems from the full-graph attention formulated in Eq. (2), which captures global information from the graph topology. However, since the mini-batch scheme replaces it with in-batch attention, its capability is potentially hindered. To compensate the information loss, more powerful embedding and encoding techniques are required to represent global graph views for the model to learn. Canonical mini-batch GT models [10, 38, 47, 51] generally utilize graph adjacency for retrieving neighborhood information and composing batch nodes. However, recent advances reveal that adjacency alone is insufficient to represent the graph topology [6]. GTs can be improved by learning node interactions beyond existing edges, and hierarchical information benefits the model in retrieving high-level graph knowledge [25, 52, 54].

Unlike existing hierarchical GTs relying on the original graph, we are thence motivated to extend the graph topology and incorporate hub labeling to represent graph information. Given a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, the graph labeling process [14] forms artificial edges $\hat{\mathcal{E}}$ to record the distance between nodes and stored as node labels. We consider this *label graph* $\hat{\mathcal{G}} = \langle \mathcal{V}, \hat{\mathcal{E}} \rangle$ containing additional directed and weighted edges for GT learning.

Building the label graph. Proposed for efficient calculation of node-pair SPD queries, representative hub labeling approaches [1, 2] build a label index $\mathcal{L}(v)$ for node v in the graph, which is a set of hub nodes u and corresponding shortest distances $b(u, v)$ between the node pairs. The label of graph nodes is said to form a *2-hop cover* if for an arbitrary node pair $u, v \in \mathcal{V}$, there exists a node $w \in \mathcal{L}(u) \cap \mathcal{L}(v)$, and the SPD can be calculated by $b(u, v) = b(u, w) + b(w, v)$, where $b(u, w)$ and $b(w, v)$ are stored in labels $\mathcal{L}(u)$ and $\mathcal{L}(v)$, respectively.

To build the labels for all nodes in the graph with efficient search space and minimal index size, the Pruned Landmark Labeling (PLL)



(a) Raw Graph **(b) Property 1.1** **(c) Property 1.2** **(d) Property 1.3**

Figure 1: Examples of properties of the label graph $\hat{\mathcal{G}}$ corresponding to the original graph $\mathcal{G}$. Number inside each node denotes its index in descending order of node degrees.

algorithm [3, 5] searches labels by a pruned Breadth First Search (BFS) for each node. The algorithm replies on a particular order of all nodes in $\mathcal{V}$. Denoting each node by a unique index $1, \cdots, n$, $u < v$ indicates that node u precedes node v in the sequence. During construction, PLL tends to regard low-index nodes as *hubs* and connecting more edges to them, while eliminating the labeling from high-index nodes. Hence, a hierarchy of nodes are intrinsically formed by labeling.

For simplicity, we assume that the original graph $\mathcal{G}$ is undirected, while properties for a directed $\mathcal{G}$ can be acquired by separately considering two label sets $\mathcal{L}_{in}$ and $\mathcal{L}_{out}$ for in- and out-edges in $\mathcal{E}$. We summarize the following three properties of the label hierarchy produced by PLL, while further explanations with examples can be found in Section 3.2:

Proposition 1 (Hierarchical Properties of Label Graph).

1. *For an edge $(u, v) \in \mathcal{E}$, there is $(v, u) \in \hat{\mathcal{E}}$ when $u < v$, and $(u, v) \in \hat{\mathcal{E}}$ when $u > v$.*
2. *For a shortest path $\mathcal{P}(u, v)$ in $\mathcal{G}$, there is $(w, v) \in \hat{\mathcal{E}}$ for each $w \in \mathcal{P}(u, v)$ satisfying $w > v$.*
3. *For a shortest path $\mathcal{P}(u, v)$ in $\mathcal{G}$, if there is $w \in \mathcal{P}(u, v)$ and $w < v$, then $(u, v) \notin \hat{\mathcal{E}}$.*

In brief, Proposition 1.1. ensures that neighboring nodes in the raw graph $\mathcal{G}$ are still connected in $\hat{\mathcal{G}}$. Proposition 1.2. and 1.3. jointly imply that a small number of hub nodes, or landmarks, naturally emerge when more shortest paths pass through these nodes, and reside in a large number of node labels during the labeling process. On the contrary, for insignificant nodes with higher indices, the pruned traversal constrains the visit to the local neighborhood and limit their label size. Thus, we reckon that the PLL process builds a hierarchy embedded in the node labels, distinguishing global hubs while preserving original adjacency.

Querying hub labels. Once graph labels are built, we adopt them in generating batches for HubGT learning, which can be formulated as the following problem for querying the index structure:

Problem Definition. *Given a node r, we aim to exploit the label graph $\hat{\mathcal{G}}$ to sample a set of nodes $\mathcal{S}(r)$ such that $(u, r) \in \hat{\mathcal{E}}$ or $(r, u) \in \hat{\mathcal{E}}$ for $u \in \mathcal{S}(r)$. In addition, for any node pair $u, v \in \mathcal{S}(r)$, we aim to acquire the shortest path distance $b(u, v)$ efficiently.*

Compared with traditional neighborhood-based batching, generating from the constructed labels preserves local neighbors while adding global hubs according to Proposition 1. This is preferable for GTs as it extends the receptive field beyond local neighbors described by graph adjacency and highlights those distant but important hubs in the whole graph for learning node interactions.

Meanwhile, it maintains the form of 1-hop sampling, rather than multi-hop operations with potentially higher overhead.

The hierarchical information is especially useful for complicated scenarios such as heterophilous graphs, where the local graph topology may be ambiguous or even misleading [27, 34, 44]. An empirical study in Figure 2 shows node homophily on the raw and label graphs. It can be observed that by adding global edges through labeling, nodes with lower scores on heterophilous graphs CHAMELEON and SQUIRREL are enhanced with more homophilous connections, while homophilous graphs CORA and CITESEER maintain the distribution. Forming a more homophilous node set benefits GT by providing similar features to learn node representation.

3.2 Label Graph Properties

In this part, we formally formulate the hierarchy in graph labels. By considering the nodes in labels as edge relationship, the wholistic label set $\mathcal{L}$ can be regarded as a derived graph $\hat{\mathcal{G}} = \langle \mathcal{V}, \hat{\mathcal{E}} \rangle$ with directed and weighted edges, namely the *label graph*. Its edge set depicts the elements in node labels computed by graph labeling, that an edge $(u, v) \in \hat{\mathcal{E}}$ if and only if $(v, \delta_r) \in \mathcal{L}(u)$, and the edge weight is exactly the distance in graph labels $\delta_r = b(u, v)$. The in- and out-neighborhoods based on edge directions are $\mathcal{N}_{in}(v) = \{u|(u, v) \in \hat{\mathcal{E}}\}$ and $\mathcal{N}_{out}(v) = \{u|(v, u) \in \hat{\mathcal{E}}\}$, respectively.

Property 1.1. Referring to Algorithm 1, when the current node is v and $v < u$, $\delta_r = 1$ holds since u is the direct neighbor of v. Hence, $(v, 1)$ is added to label $\mathcal{L}(u)$ at this round, which is equivalent to adding edge (u, v) to $\hat{\mathcal{E}}$. Similarly, $(v, u) \in \hat{\mathcal{E}}$ holds when $v > u$. For example, the edge $(1, 4)$ in Figure 1(a) is represented by the directed edge $(4, 1)$ in Figure 1(b). Property 1.1. implies that $\mathcal{N}(v) \subset \mathcal{N}_{in}(v) \cup \mathcal{N}_{out}(v)$, i.e., the neighborhood of the original graph is also included in the label graph, and is further separated into two sets according to the relative order of neighboring nodes.

Property 1.2. [5] proves that there is $v \in \mathcal{L}(w)$ for $w \in \mathcal{P}(u, v)$ and $w > v$. Therefore, considering shortest paths starting with node v of a small index, i.e., v being a "landmark" node, then succeeding nodes $w > v$ in the path are connected to v in $\hat{\mathcal{G}}$. In Figure 1(a), the shortest path between $(1, 5)$ passing node 2 results in edges $(2, 1)$ and $(5, 1)$ in Figure 1(c), since nodes 2 and 5 are in the path and their indices are larger than node 1. When the order is determined by node degree, high-degree nodes appear in shortest paths more frequently, and consequently link to a majority of nodes, including those long-tailed low-degree nodes in $\hat{\mathcal{G}}$.

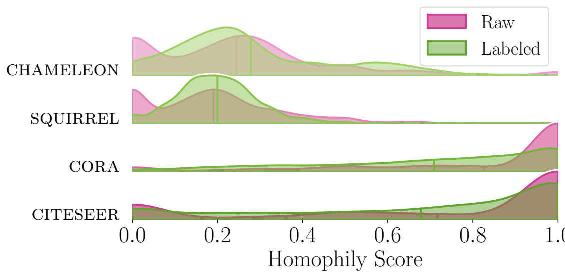


Figure 2: Relative distribution of homophily scores between nodes in the original and label graphs. A higher score implies a higher ratio of similar nodes in the neighborhood.

Property 1.3. According to the property of shortest path, there is $b(u, v) = b(u, w) + b(w, v)$. Hence, the condition of line 8 in Algorithm 1 is not met at the v-th round when visiting w. In other words, the traversal from v is pruned at the preceding node w. By this means, the in-neighborhood $\mathcal{N}_{in}(v)$ is limited in the local subgraph with shortest paths ending at landmarks. As shown in Figure 1(d), the shortest path between $(5, 6)$ passes node 1, indicating that $(5, 6)$ are not directly connected since their distance can be acquired by edges $(5, 1)$ and $(6, 1)$. As a consequence, the neighborhood of node 5 in $\hat{\mathcal{G}}$ is constrained by nodes 1 and 2, preventing connections to more distant nodes such as 3 or 6.

Summarizing Properties 1.1., 2., and 3., the label graph preserves neighboring connections of the original graph, while establishing more connections to a minority set of global nodes as landmark. The hierarchy is built so that long-tailed nodes with high indices are usually located in local substructures separated by landmarks.

3.3 HubGT Indexing

Design Goals. The stated problem characterizes the query requirements for designing our specific HubGT algorithms. In particular, we embrace the decoupling principle in consideration of scalability, separating the GT workflow into *precomputation* and *training* phases, as shown in Figure 3. Since graph labels are deterministic, they can be computed in an individual indexing stage beforehand. The training stage, on the other hand, focuses on variable operations, including querying neighboring nodes and SPD, to learn model weights through repetitive epochs.

As analyzed in Table 1, GT training usually dominates the learning overhead and determines the deployment budget in practice. From the aspect of graph computation, a training epoch sends queries from every nodes in the graph. In comparison, the index is computed only once and used throughout training. Therefore, our primary design objective is to minimize the repetitive overhead of querying nodes and distances during training. The secondary goal is to achieve a reasonable time and space overhead for indexing. To achieve this, we propose our novel HubGT indexing and corresponding querying algorithms encompassing three levels of hierarchies, which are respectively introduced as follows.

Hierarchy-1: Local hub labeling. Examining the problem definition in Section 3.1, we observe that, unlike the conventional SPD query on an arbitrary node pair in the graph, the query for $b(u, v)$ in HubGT always orients an intermediary node r. In other words, u, v are 2-hop neighbors in the label graph $\hat{\mathcal{G}}$ disregarding edge directions. To leverage this relevance, we broaden the derivation of 2-hop labeling in [5], that the distance candidate orienting $b_r(u, v)$ can be acquired by utilizing the connectivity with r:

$$b_r^{(1)}(u, v) = b(u, r) + b(r, v) - \delta_r(u, v), \tag{3}$$

where $\delta_r(u, v) = 2, 1,$ or 0 depending on the relative position between (u, r) and (v, r) in $\mathcal{G}$. Denote $\mathcal{M}_r^i(v) = \{w \in \mathcal{N}(r) \mid b(r, v) - b(w, v) = i\}$, $i = -1, 0,$ or 1, where $\mathcal{N}(r) = \{w|(w, r) \in \mathcal{E}\}$ is the neighborhood orienting r in $\mathcal{G}$. The node sets $\mathcal{M}_r^i(v)$ of different integers i effectively characterize the relative position of the node of interest v with respect to the intermediary node r. When $\mathcal{M}_r^1(u) \cap \mathcal{M}_r^1(v) \neq \varnothing$, there is $\delta_r(u, v) = 2$; when $\mathcal{M}_r^0(u) \cap \mathcal{M}_r^1(v) \neq \varnothing$ or $\mathcal{M}_r^1(u) \cap \mathcal{M}_r^0(v) \neq \varnothing$, there is $\delta_r(u, v) = 1$; otherwise $\delta_r(u, v) = 0$.

We develop Algorithm 1 as the first level of hierarchy (H-1) based on PLL and Eq. (3). During the pruned BFS, we record the neighbor sets $\mathcal{M}_r^1(v)$ and $\mathcal{M}_r^0(v)$ for nodes added to labels $\mathcal{L}(u)$ along with their indices and distances. Additionally, we also maintain an inverse label set $\mathcal{L}'(v)$ such that $u \in \mathcal{L}'(v)$ if and only if $v \in \mathcal{L}(u)$. Therefore, each element in labels $\mathcal{L}(v)$ or $\mathcal{L}'(v)$ is a quadruple $(u, b(u,v), \mathcal{M}_v^1(u), \mathcal{M}_v^0(u))$. Eq. (3) of distance calculation and set intersection can be boosted by bit-parallel operations. $\mathcal{M}_r^i(v)$ can also be stored bit-wise in a word. Hence, constructing the H-1 index shares the same overhead with PLL, while offering the extended distance information based on local computation orienting hubs.

In specific, we also explicitly impose a maximum capacity s for each node label $\mathcal{L}(v)$ in Algorithm 1. This is because at most s neighbors are required to form the batch in our problem. Notably, the algorithm is agnostic to the search order. In this work, we follow [50] to adopt the descending order of node degrees as it can be efficiently acquired and offers decent performance. While other orders such as betweenness centrality are adopted for labeling [36, 41], they nonetheless incur additional computational overhead.

Hierarchy-0: 2-hop pair caching. Although the H-1 query Eq. (3) produces the shortest distance $b(u,v)$ when $b_r^{(1)}(u,v) \le b(u,v) + \delta_r(u,v)$, it is possible that the actual shortest path $\mathcal{P}(u,v)$ does not pass through r or its neighbors. In this case, the query falls back to the classic node labeling scheme:

$$b^{(0)}(u,v) = \min_{w \in \mathcal{L}(u) \cap \mathcal{L}(v)} \{b(u,w) + b(w,v)\}, \tag{4}$$

where labels $\mathcal{L}$ are computed by Algorithm 1. Note that $b(v,v) = 0$.

We regard the 2-hop labeling scheme as the fundamental hierarchy (H-0) as it guarantees answer to any queries. However, calculation by Eq. (4) results in a complexity of $O(|\mathcal{L}(u)| + |\mathcal{L}(v)|) = O(s)$, which is still not satisfying under the repetitive querying scenario in GT training. To further improve query speed, we choose to cache the frequently queried 2-hop shortest distances $b(u,v)$ and index them by the end node as $\mathcal{I}_v[u]$ for $u < v$. By employing a suitable data structure such as a hash map for each v, checking the existence and acquiring $\mathcal{I}_v[u]$ for a given node pair can be completed in $O(1)$.

Algorithm 1: HubGT H-1 Indexing

```
Input: Graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, Max neighbor size $s$
Output: H-1 Labels $\mathcal{L}, \mathcal{L}'$
1  Sort $\mathcal{V}$ based on degree $d(v)$
2  $\mathcal{L}(v) \leftarrow \varnothing$, $\mathcal{L}'(v) \leftarrow \varnothing$ for all $v \in \mathcal{V}$
3  for $r = 1$ to $n$ do
4      $(q(v), \mathcal{M}_r^1(v), \mathcal{M}_r^0(v)) \leftarrow (\infty, \varnothing, \varnothing)$ for all $v \in \mathcal{V}$
5      $(q(v), \mathcal{M}_r^1(v), \mathcal{M}_r^0(v)) \leftarrow (1, \{v\}, \varnothing)$ for all $v \in \mathcal{N}(r)$
6      Queue $Q \leftarrow \{(r, 0)\}$, $(q(r), \mathcal{M}_r^1(r), \mathcal{M}_r^0(r)) \leftarrow (0, \varnothing, \varnothing)$
7      while $Q \neq \varnothing$ do
8          $Q^1 \leftarrow \varnothing$, $Q^0 \leftarrow \varnothing$
9          Pop the first element $(v, b(v,r))$ from $Q$
10         Get $b^{(0)}(v,r)$ from Eq. (4) with existing labels
11         if $b(v,r) < b^{(0)}(v,r)$ and $|\mathcal{L}(v)| < s$ then
12             $\mathcal{L}(v) \leftarrow \mathcal{L}(v) \cup (r, b(v,r))$
13             $\mathcal{L}'(r) \leftarrow \mathcal{L}'(r) \cup (v, b(v,r))$
14             for all $u \in \mathcal{N}(v)$ such that $u > r$ do
15                 if $q(u) > q(v)$ then
16                     Push $(u, b(v,r) + 1)$ to the end of $Q$
17                     $Q^1 \leftarrow Q^1 \cup \{(u,v)\}$, $q(u) \leftarrow q(v) + 1$
18                 else if $q(u) = q(v)$ then
19                     $Q^0 \leftarrow Q^0 \cup \{(u,v)\}$
20         for all $(u,v) \in Q^0$ do
21             $\mathcal{M}_r^0(u) \leftarrow \mathcal{M}_r^0(u) \cup \mathcal{M}_r^1(v)$
22         for all $(u,v) \in Q^1$ do
23             $\mathcal{M}_r^1(u) \leftarrow \mathcal{M}_r^1(u) \cup \mathcal{M}_r^1(v)$
24             $\mathcal{M}_r^0(u) \leftarrow \mathcal{M}_r^0(u) \cup \mathcal{M}_r^0(v)$
25 return $\mathcal{L}(v), \mathcal{L}'(v)$ for all $v \in \mathcal{V}$
```

The precomputation for building the H-0 index is in Algorithm 2. Examining Proposition 1 and Eq. (4), it can be inferred that the intermediary node presents on the shortest path $r \in \mathcal{P}(u,v)$ if and only if $r \in \mathcal{L}(u) \cap \mathcal{L}(v)$. We hence equivalently rearrange the traversal order of 2-hop pairs $u \in \mathcal{L}'(r)$, $r \in \mathcal{L}(v)$ for all possible presence of shortest paths, where $\mathcal{L}, \mathcal{L}'$ are produced by Algorithm 1. The search and $\mathcal{I}_v$ construction for each node v can be performed in parallel since they are mutually independent.

In Algorithm 2, we utilize two structures $\mathcal{I}_v^{(0)}$ and $\mathcal{I}_v^{(1)}$ to respectively record the distances acquired by Eq. (4) and Eq. (3), and save the distance to index $\mathcal{I}_v$ only if there exists a node r that cannot calculate the shortest distance using the H-1 index. In this way, we effectively constrain the H-0 index size, and ensure that it only caches the cases where H-1 may not produce the shortest distance.

Hierarchy-2: Global hub labeling. H-1 index presents the idea of utilizing the relative position orienting a given hub r to compute distances of 2-hop pairs present in the query. It can be further extended to some *global* hubs by indexing their labels as another level of hierarchy (H-2), which corresponds to the bit-parallel BFS in [5]. More specifically, we select a small set of nodes with low indices and perform BFS without pruning following Algorithm 1. For each global node t, the label $(b(v,t), \mathcal{M}_t^1(v), \mathcal{M}_t^0(v))$ is computed and stored for all $v \in \mathcal{V}$. Then, the distance of an arbitrary node pair $b_t^{(2)}(u,v)$ can be similarly computed by Eq. (3) orienting t.

In our implementation, the H-2 construction is performed prior to H-1 and H-2, offering an additional condition for BFS pruning and cache saving. As demonstrated in [5], the global index design is advantageous in reducing the overall label size and facilitating a faster indexing time. It also benefits from bit-parallel processing and fast SPD computation similar to H-1.

3.4 HubGT Querying

After building the index H-2, H-1, and H-0 successively, querying label graph neighbors $\mathcal{S}(r)$ and their distances as in Section 3.1 can

Algorithm 2: HubGT H-0 Indexing

```
Input: Graph labels $\mathcal{L}, \mathcal{L}'$
Output: H-0 Index $\mathcal{I}$
1  for $v = 1$ to $n$ do                                  ▷ [in parallel]
2      $\mathcal{I}_v^{(0)}[u] \leftarrow \infty$, $\mathcal{I}_v^{(1)}[u] \leftarrow -\infty$ for all $u \in \mathcal{V}$
3      for all $r \in \mathcal{L}(v)$ do
4          for all $u \in \mathcal{L}'(r)$ such that $u < v$ do
5              Get $b_r^{(1)}(v,u)$ from Eq. (3)
6              if $b_r^{(1)}(v,u) > \mathcal{I}_v^{(1)}[u]$ then $\mathcal{I}_v^{(1)}[u] \leftarrow b_r^{(1)}(v,u)$
7              $b^{(0)}(v,u) \leftarrow b(v,r) + b(r,u)$
8              if $b^{(0)}(v,u) < \mathcal{I}_v^{(0)}[u]$ then $\mathcal{I}_v^{(0)}[u] \leftarrow b^{(0)}(v,u)$
9      for all $u \in \mathcal{V}$ such that $\mathcal{I}_v^{(0)}[u] \neq \infty$ do
10         if $\mathcal{I}_v^{(0)}[u] < \mathcal{I}_v^{(1)}[u]$ and $|\mathcal{I}_v| < s^2$ then
11             $\mathcal{I}_v[u] \leftarrow \mathcal{I}_v^{(0)}[u]$
12 return $\mathcal{I}_v$ for all $v \in \mathcal{V}$
```

be achieved solely on the $\mathcal{L}, \mathcal{L}', \mathcal{I}$ without resorting to the graph structure. In Algorithm 3, we showcase the sampling procedure given an ego node r and respective sample sizes s_{out} and s_{in} for out- and in-connections stored in $\mathcal{L}(r)$ and $\mathcal{L}'(r)$, respectively. The node-pair distance inside $\mathcal{S}(r)$ is presented as a symmetric matrix $\boldsymbol{B}_r$, and its entry value $\boldsymbol{B}_r[u,v] = \boldsymbol{B}_r[v,u] = b(u,v)$.

The distance query on (u,v) consecutively accesses H-0, H-1, and H-2 indices in Algorithm 3. If the H-0 distance $\mathcal{I}_v[u]$ exists, it indicates the shortest distance according to Algorithm 2. Otherwise, the distance is either achieved by the local or global hub labeling following Eq. (3) orienting a particular intermediary node r or t. Notably, queries regarding the ego node $\mathcal{S}(r)$ can be performed by only accessing the index of r without referring to others, which offers better memory locality and runtime efficiency.

Complexity Analysis. We respectively investigate the time and memory complexity of each indexing and querying stages. For H-1 and H-2 indexing, the algorithm completes the traversal of all nodes in $O(ns+ms)$ time, considering the label size $|\mathcal{L}(v)| \leq s$. The total index size is therefore bounded by $O(ns)$. For H-0 labeling in Algorithm 2, the computational overhead is $O(nss')$, where s' is the average size of $\mathcal{L}'$. Empirically, we enforce the index size to be less than s^2 for every $\mathcal{I}_v$. In summary, the time and memory complexities for the three-level indexing are $O(ns^2+ms)$ and $O(ns^2)$, respectively. We particularly highlight that the empirical label sizes observed in experiments for both H-0 and H-1 are substantially smaller than the theoretical bound, thanks to the hierarchy that effectively reduces redundant information.

Querying one node pair distance by Algorithm 3 can be achieved in $O(1)$ time when bit parallel and accessing hash map are both $O(1)$ operations. By selecting sample sizes such that $s = s_{in}+s_{out}+1$, the subgraph size for each query node is $|\mathcal{S}(r)| = s$. In consequence, querying every node as $r \in \mathcal{V}$ in each training iteration, with each query node possessing at most s^2 pair-wise distance computations, entails $O(ns^2)$ overhead. Compared to the conventional full-graph SPD query under $O(n^3)$ complexity, this is a significant improvement benefiting model scalability.

Comparison with Traditional Hub Labeling. We highlight that HubGT and canonical hub labeling address distinct query scenarios and design goals, leading to different performance of index construction and SPD querying. In HubGT, we prioritize querying with $O(1)$ overhead and local label access, in order to prevent the excessive queries from blocking GT training. To this end, we adopt the H-0 cache with additional precomputation and index size for fast distance access, while leveraging the neighboring property on label graph to construct H-1 index. Contrarily, classic hierarchical labeling approaches [5, 36] are designed for arbitrary node-pair queries, which is not applicable to the intermediary node technique in our H-1 local labeling, and entails $O(s)$ query time.

Algorithm 3: HubGT Query for node r

```
Input: Index $\mathcal{L}, \mathcal{L}', \mathcal{I}$, Sample sizes $s_{in}, s_{out}$
Output: Sampled subgraph nodes $\mathcal{S}(r)$ and SPD matrix $\boldsymbol{B}_r$
1  $\mathcal{S}(r) \leftarrow \{r\}$
2  Sample $s_{out}$ nodes from $\mathcal{L}(r)$ into $\mathcal{S}(r)$
3  Sample $s_{in}$ nodes from $\mathcal{L}'(r)$ into $\mathcal{S}(r)$
4  for all $(u,v)$ such that $u \in \mathcal{S}(r), v \in \mathcal{S}(r), u < v$ do
5      if $\mathcal{I}_v[u]$ exists then
6          $b(u,v) \leftarrow \mathcal{I}_v[u]$
7      else
8          Get $b_r^{(1)}(u,v)$ from Eq. (3)
9          Get $b_t^{(2)}(u,v)$ from Eq. (3) for all global $t$
10         $b(u,v) \leftarrow \min\{b_r^{(1)}(u,v), b_t^{(2)}(u,v)\}$
11     $\boldsymbol{B}_r[u,v] \leftarrow b(u,v), \boldsymbol{B}_r[v,u] \leftarrow b(u,v)$
12 return $\mathcal{S}(r)$ and $\boldsymbol{B}_r$
```

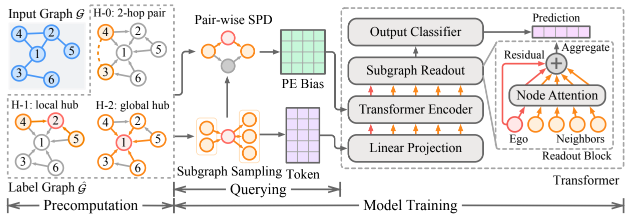


Figure 3: HubGT framework including precomputation and training. The precomputation stage processes the input graph and constructs the label index. During training, subgraph nodes and SPDs are queried from the index and applied as different Transformer inputs. Querying on CPU and training on GPU are pipelined and conducted simultaneously.

4 HubGT MODEL DESIGN

To efficiently exploit the label graph in HubGT, in this section, we elaborate our end-to-end design representing graph hierarchy as both graph embeddings and positional encoding in GT learning. Figure 3 illustrates the overview of the HubGT pipeline.

4.1 Subgraph for Node Embeddings

In mini-batch GT training, both graph embeddings and positional encoding are essential to represent graph information for model learning. Each embedding is also known as a *token*, which is the minimum unit for model representation. Alternatively, the encoding $\boldsymbol{P}$ is used in attention layers in Eq. (2) for depicting pair-wise relationship within the subgraph. Leveraging the label graph hierarchy in HubGT benefits both of these two inputs: (1) The labels provide expressive graph information for generating *embeddings*, encompassing both local neighbors and global hubs. (2) Node-pair SPD can be efficiently calculated, which is then used as positional *encoding* for evaluating relevance between nodes.

To generate embeddings of a node v in HubGT, we leverage the subgraph hierarchy in the label graph, i.e., neighboring nodes in labels $\mathcal{L}(v)$ and $\mathcal{L}'(v)$. Considering that the neighborhood size is variable, we convert it into a fixed-length token $\mathcal{S}(v)$ by sampling to align the GT architecture. When $\mathcal{S}(v)$ contains neighbors from $\mathcal{L}(v)$ and $\mathcal{L}'(v)$, as well as the ego node v itself, its length is $s = s_{in}+s_{out}+1$, which justifies the problem definition in Section 3.1. The node embedding is generated by concatenating the raw attributes $\boldsymbol{X}[u]$ of subgraph nodes $u \in \mathcal{S}(v)$, denoted as $\boldsymbol{X}[\mathcal{S}(v)]$.

The relative values of hyperparameters s_{in} and s_{out} can be used to balance the ratio between in-neighbors and out-neighbors in $\hat{\mathcal{G}}$, which correspond to local long-tailed nodes and distant landmark nodes in $\mathcal{G}$, respectively. Compared to canonical GT tokens that represent the graph node in the context of the full graph, HubGT only relies on a small but informative subgraph of fixed size s. When the graph scales up, HubGT enjoys better scalability as its token size does not increase with the graph size.

Global Hubs. Section 3.3 signifies a set of global nodes used for H-2 labeling. For generating node embeddings, we further leverage these nodes as global hubs, that we consider these hubs connected to every nodes $v \in \mathcal{V}$ and can be similarly sampled during token generation. In HubGT training, we set their attributes to be learnable along with model weights. This scheme actually generalizes the virtual node utilized in [51] to a set of hubs. The learnable embeddings are able to aggregate information from connected nodes throughout learning, which eventually offers an representation of the graph in a higher level of hierarchy, and benefits GT for retrieving graph-level information.

4.2 Distance for Positional Encoding

For positional encoding, we specifically choose to use SPD, which depicts relative node position in the graph by pair-wise distances. In the context of mini-batch GT training, SPD is advantageous in providing dense representation for every node pairs regardless of their position in the graph, and can be efficiently acquired without auxiliary memory overhead. In comparison, graph proximity and Laplacian encoding commonly used for mini-batch GTs only possess meaningful values for local nodes within a limited hops, and are usually too sparse in an RS batch with most entries as zeros, which limits its expressivity in depicting global relationships.

Despite the strength of SPD encoding, it has not been employed for large-scale GTs due to the impractical $O(n^3)$ overhead in naive calculation. Thanks to the efficient graph labeling of HubGT, we are able to efficiently acquire SPD inside subgraphs by performing queries on the index. For each node r, Algorithm 3 calculates the distances $\boldsymbol{B}_r$ of all node pairs inside the token $\mathcal{S}(r)$ under $O(s^2)$ complexity. A learnable embedding scheme $f_B : \mathbb{N} \to \mathbb{R}$ is then employed to map the distance of each entry in $\boldsymbol{B}_r$ to the attention bias $\boldsymbol{P}$ used in Eq. (2), that $\boldsymbol{P}[u,v] = f_B(\boldsymbol{B}_r[u,v])$.

In implementation, querying a batch of ego nodes by Algorithm 3 can be conducted in parallel. The SPD query on the CPU can be pipelined with GT training on the GPU, so that training with the current batch of embeddings and encoding can be performed simultaneously with fetching and loading the data of the next batch. Thanks to the fast query design, the HubGT workflow ensures that accessing the index does not block the computation-intensive GT training, allowing seamless training that can potentially be enhanced by further acceleration techniques developed for the Transformer architecture.

4.3 Overall Architecture

HubGT adapts the scalable GT architecture [51, 52] incorporating subgraph precomputation and mini-batch training. The token features of each node v are learned from the subgraph embeddings as $\boldsymbol{H}[v] = \text{MLP}_X(\boldsymbol{X}[\mathcal{S}(v)])$, where $\text{MLP}_X : \mathbb{R}^{s\times F_0} \to \mathbb{R}^{s\times F}$ with hidden dimension F. Then, L Transformer layers characterized by Eqs. (1) and (2) are applied to predict the node representation $\boldsymbol{H}[v]$ with positional encoding $\boldsymbol{P}$. Lastly, we introduce a readout block to calculate attention within the token $\mathcal{S}(v)$ to aggregate the output of each ego node:

$$Z[v] = \text{MLP}_Z\Big(\boldsymbol{H}^{(L)}[v] + \sum_{u\in\mathcal{S}(v)} \alpha_u \boldsymbol{H}^{(L)}[u]\Big),$$
$$\text{where} \quad \alpha_u = \frac{\exp\big((\boldsymbol{H}^{(L)}[v] \| \boldsymbol{H}^{(L)}[u])\boldsymbol{W}_E\big)}{\sum_{u\in\mathcal{S}(v)} \exp\big((\boldsymbol{H}^{(L)}[v] \| \boldsymbol{H}^{(L)}[u])\boldsymbol{W}_E\big)}, \tag{5}$$

and MLP_Z is the output classifier.

Data Batching and Transferring. Remarkably, HubGT inputs of embedding and encoding can be manipulated in-place on $\boldsymbol{X}$ and $\boldsymbol{B}_r$, and no graph-scale computation is required during learning iterations. Therefore, mini-batch training can be easily implemented by randomly sampling batches of ego nodes, and only strides of $\boldsymbol{X}$ and batches of $\boldsymbol{B}_r$ are loaded onto GPUs.

Complexity Analysis. During model training, one epoch of L-layer feature transformation on all nodes entails $O(LnF)$ complexity, while positional encoding is performed under $O(ns^2)$. The RAM footprint is $O(ns^2)$ and $O(nsF)$ for sampled tokens and features, respectively. For mini-batch training with batch size n_b, the VRAM overhead on GPU for a batch of node representations and bias matrices is $O(Ln_bF)$ and n_bs^2, respectively. It can be observed that the GPU memory footprint is determined only by batch size and is independent of the graph scale, ensuring favorable scalability.

5 EXPERIMENTS

We comprehensively evaluate the performance of HubGT with a wide range of datasets and baselines. In Section 5.2, we highlight the model efficiency throughout learning phases as well as its effectiveness under both homophily and heterophily. Sections 5.3 and 5.4 provides in-depth insights into the HubGT design in exploiting graph hierarchy.

5.1 Experimental Settings

Tasks and Datasets. We focus on the node classification task on 14 benchmark datasets covering both homophily [19, 39, 40] and heterophily [32, 35]. Compared to conventional graph learning tasks used in GT studies, this task requires learning on large single graphs, which is suitable for assessing model scalability. We follow common data processing and evaluation protocols as detailed in [31]. Evaluation is conducted on a server with 32 Intel Xeon CPUs (2.4GHz), an Nvidia A30 GPU (24GB memory), and 512GB RAM.

Baselines. Since the scope of this work lies in the efficacy and efficiency enhancement of the GT architecture, we primarily compare against leading scalable Graph Transformer models with attention-based layers and mini-batch capability. Methods including DIFFormer [46] and PolyNormer [15] are considered as kernel-based approaches. NAGphormer [10], GOAT [25], HSGT [54], and ANS-GT [52] stand for hierarchical GTs. An state-of-the-art message-passing GNN SGFormer [48] is also included for comparison.

Evaluation Metrics. We use ROC AUC as the efficacy metric on TOLOKERS and classification accuracy on the other datasets. For efficiency evaluation, we notice that there is limited consensus due to the great variety in GT training schemes. Therefore, we attempt to employ a comprehensive evaluation considering processing times of different learning phases for a fair comparison. Model speed is

Table 2: Effectiveness and efficiency results on large-scale graph datasets, while full evaluation are in our tech report [31]. "Pre.", "Epoch", and "Infer" are precomputation, training epoch, and inference time (in seconds), respectively. "OOM" implies that the model encounters the out-of-memory error. "TLE" means the learning process exceeds the time limit of 24 hours before convergence. Respective results of the first and second best performances are marked in bold and underlined fonts.

Homophilous	PHYSICS				OGBN-ARXIV				REDDIT				OGBN-MAG			
Scale	34, 493 / 495, 924				169, 343 / 2, 315, 598				232, 965 / 114, 615, 892				736, 389 / 10, 792, 672			
Metrics	Pre.	Epoch	Infer	Acc	Pre.	Epoch	Infer	Acc	Pre.	Epoch	Infer	Acc	Pre.	Epoch	Infer	Acc
DIFFormer*	-	1.7	3.8	96.10±0.11	-	0.89	4.1	55.90±8.23	-	2.4	21	94.96±0.37	-	1.7	9.7	31.13±0.48
PolyNormer*	-	0.76	2.4	**96.59**±0.16	-	0.83	11	**73.24**±0.13	-	5.3	246	**96.64**±0.07	-	20	992	32.42±0.15
SGFormer*	-	0.52	2.6	96.33±0.29	-	0.82	1.9	72.55±0.28	-	2.2	21	95.63±0.29	-	1.7	5.1	33.47±0.61
NAGphormer	33	8.4	2.4	96.52±0.24	18	4.4	2.2	67.85±0.17	280	3.2	2.1	95.77±0.08	89	10.3	2.2	33.23±0.06
ANS-GT	2203	63	35	96.31±0.28	16205	109	2.7	71.06±0.48		(OOM)				(OOM)		
GOAT	45	14	12	96.24±0.15	1823	48	61	69.66±0.73	628	141	104	(TLE)	2673	116	102	(TLE)
HSGT*	12	41	62	96.05±0.50	16	475	142	68.30±0.32	614	453	482	(TLE)	182	582	629	(TLE)
HubGT (ours)	2.0	2.9	0.31	96.38±0.25	30	9.3	1.3	69.17±0.33	192	21	1.6	94.39±0.06	523	62	1.2	**33.74**±0.24

Heterophilous	PENN94				GENIUS				TWITCH-GAMER				POKEC			
Scale	41, 554 / 2, 724, 458				421, 961 / 1, 845, 736				168, 114 / 13, 595, 114				1, 632, 803 / 44, 603, 928			
Metrics	Pre.	Epoch	Infer	Acc	Pre.	Epoch	Infer	Acc	Pre.	Epoch	Infer	Acc	Pre.	Epoch	Infer	Acc
DIFFormer*	-	0.53	0.65	61.77±3.41	-	0.77	5.5	84.52±0.36	-	0.61	5.1	60.81±0.44	-	4.6	15	73.89±0.35
PolyNormer*	-	0.58	18.4	**79.87**±0.06	-	0.77	28	85.64±0.52	-	1.45	89	64.72±0.65	-	4.2	67	**81.03**±0.08
SGFormer*	-	0.95	0.55	77.52±0.56	-	0.72	2.4	85.01±0.25	-	0.38	3.3	65.93±0.15	-	4.4	29	73.13±0.16
NAGphormer	237	6.1	2.1	74.45±0.60	38	5.4	1.0	83.88±0.13	16	1.9	2.4	61.92±0.19	70	16.1	3.1	73.06±0.05
ANS-GT	3889	42	4.9	67.76±1.32	34092	37	5.0	67.76±1.32	12924	19	6.7	61.55±0.45		(OOM)		
GOAT	1332	33	18	71.42±0.44	2664	28	39	80.12±2.32	3348	37	63	61.38±0.83	3855	760	804	(TLE)
HSGT*	12	115	110	67.77±0.27	21	98	114	84.03±0.24	68	235	253	61.60±0.09	551	1420	1557	(TLE)
HubGT (ours)	7.3	3.4	0.29	78.13±0.43	125	23	2.5	**89.68**±0.48	49	12	1.2	**67.03**±2.17	5422	99	13	76.96±0.44

* Inference of these models is performed on the CPU in a full-batch manner due to their requirement of the whole graph.

represented by the average training time per epoch and the inference time on the testing set. For models with graph precomputation, the time for this process is separately recorded.

Dataset Details. Table 3 displays the scales and heterophily status of graph datasets utilized in our work. Undirected edges twice in the table. CHAMELEON and SQUIRREL are the filtered version from [35], while OGBN-MAG is the homogeneous variant. We employ 60/20/20 random data splitting percentages for training, validation, and testing sets, respectively, except for OGBN-MAG, where the original split is used. Regarding efficacy metrics, ROC AUC is used on TOLOKERS following the original settings, and accuracy is used for the rest.

Hyperparameters. Parameters regarding the precomputation stage for graph structures are discussed in Section 5.3. For subgraph sampling, we perform parameter search for relative ratio of in/out neighbors represented by s_{out} in rage $[0, 48]$.

For network architectural hyperparameters, we use $L = 4$ Transformer layers with $N_H = 8$ heads and $F = 128$ hidden dimension for our HubGT model across all experiments. The dropout rates for inputs (features and bias) and intermediate representation are 0.1 and 0.5, respectively. The AdamW optimizer is used with a learning rate of 10^{-4}. The model is trained with 300 epochs with early stopping. Since baseline GTs employ different batching strategies, it is difficult to unify the batch size across all models. We set the batch size to the largest value in the available range without incurring out of memory exception on our 24GB GPU, intending for a fair efficiency evaluation considering both learning speed and space.

Table 3: Statistics of graph datasets. f and N_c are the numbers of input attributes and label classes, respectively. "Train" is the portion of training set w.r.t. labeled nodes.

Hetero.	Dataset	Nodes n	Edges m	F	N_c	Train
Homo.	CHAMELEON	890	17, 708	2325	5	60%
	SQUIRREL	2, 223	93, 996	2089	5	60%
	TOLOKERS	11, 758	1, 038, 000	10	2	60%
	PENN94	41, 554	2, 724, 458	4814	2	60%
	GENIUS	421, 961	1, 845, 736	12	2	60%
	TWITCH-GAMER	168, 114	13, 595, 114	7	2	60%
	POKEC	1, 632, 803	30, 622, 564	65	2	60%
Hetero.	CORA	2, 708	10, 556	1433	7	60%
	CITESEER	3, 279	9, 104	3703	6	60%
	PUBMED	19, 717	88, 648	500	3	60%
	PHYSICS	34, 493	495, 924	8415	5	60%
	OGBN-ARXIV	169, 343	2, 315, 598	128	40	54%
	REDDIT	232, 965	114, 615, 892	602	41	60%
	OGBN-MAG	736, 389	10, 792, 672	128	349	85%

5.2 Performance Comparison

Table 2 and Table 5 presents the efficacy and efficiency evaluation results on 8 large-scale graphs and 6 small-scale graphs, respectively. As an overview, HubGT demonstrates fast computation speed and favorable mini-batch scalability throughout the learning process and is applicable to million-scale graphs. It also achieves top-tier accuracy superior to baseline models on 11 out of 14 datasets.

Efficiency. Benefiting from the decoupled architecture, HubGT is powerful in achieving competitive speed with existing efficiency-oriented GTs. It consistently showcases the fastest inference speed, since Section 4.3 elaborates that label querying can be pipelined in asynchronous execution, and the process is as simple as Transformer operations without the interference of graph computation.

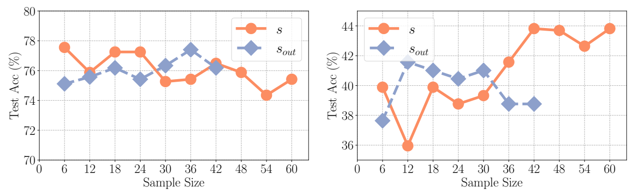
In comparison to other hierarchical GTs, HubGT excels with the fastest training and overall learning time. Specifically, its precomputation is 250-1000× faster than ANS-GT, which is also relatively fast in model training and inference. Aligned with our complexity analysis in Section 2, the overhead of HubGT indexing is mainly relevant to the node size n and is less affected by m and F compared to precomputation in other methods, which ensures its efficiency on denser graphs such as REDDIT and PENN94. Baselines models employing graph-altered learning schemes including DIFFormer, PolyNormer, and SGFormer are empirically fast in training due to the simplified model architecture. However, it is noticeable that their inference reply on the full-graph structure and can be only performed on CPU without GPU computation, resulting in slower and less scalable performance.

We additionally note that while some models claim to be applicable to million-scale graphs as shown in Table 1, they exhibit weaker scalability in our settings mainly due to the excessive training time, which prevents the model from converging even over a long time span. In particular, ANS-GT demands a high memory footprint for storing and adjusting its subgraphs, which exceeds the memory limit of our platform for the largest graphs.

Memory Footprint. In modern computing platforms, GPU memory is usually highly constrained and becomes the scalability bottleneck for the resource-intensive graph learning. HubGT exhibits efficient utilization of GPU for training with larger batch sizes while avoiding the out-of-memory issue. In comparison, drawbacks in several model designs prevent them from efficiently performing GPU computation, which stems from the adoption of graph operations. Notably, kernel-based models require full graph message-passing in their inference stage, which is largely prohibitive on GPUs and can only be conducted on CPUs. HSGT faces the similar issue caused by its graph coarsening module. We note that these solutions are less scalable and hinder the GPU utilization during training.

Prediction Efficacy. HubGT successfully achieves top or comparable accuracy on evaluated datasets in Table 2 and [31], with significant accuracy improvement on several graphs such as CHAMELEON and TWITCH-GAMER. We attribute the performance gain to the application of the label graph hierarchy and SPD positional encoding in HubGT, which offers global information and effectively addresses the heterophily issue of certain graphs as analyzed in Section 3.1. The label sampling scheme also facilitates learning on hierarchy throughout queries under a controlled overhead. Since the label graph also preserves edges in the raw graph, the performance of HubGT is usually not lower than learning on the latter.

In comparison, baseline methods without hierarchical graph designs, including DIFFormer, NAGphormer, and HSGT, perform relatively worse especially under heterophily. This is because their models tend to rely on the raw adjacency or even promote it with higher modularity. As a consequence, node connections retrieved by GT attention modules are restrained in the local neighborhood and hardly produce accurate classifications. On the other hand, while PolyNormer achieves remarkable accuracy on several graphs thanks to its strong expressivity, its performance is largely suboptimal on small homophilous graphs as we further evaluated in [31].



(a) CITESEER **(b) CHAMELEON**

Figure 4: Effect of sample sizes on different datasets.

5.3 Effect of Hyperparameters

We then study the effectiveness of the label graph hierarchy in HubGT featuring the subgraph generation process in Figure 4, which displays the impact of sample sizes s and s_{out} corresponding to Algorithm 3. Regarding the total subgraph size s, it can be observed that a reasonably large s is essential for effectively representing graph labels and achieving stable accuracy. In the main experiments, we uniformly adopt a constant $s = 48$ token size across all datasets, as it is large enough to cover the neighborhood of most nodes while maintaining computational efficiency. As a reference, the actual average H-1 index sizes of $|\mathcal{L}(v) + \mathcal{L}'(v)|$ among all nodes are 40.6 on PHYSICS and 47.7 on PENN94, while the average H-0 sizes $|\mathcal{I}(v)|$ are 230.9 and 440.1, respectively. Both are significantly smaller than the theoretical bounds of $2s$ and s^2, which validates the efficiency of our three-level hierarchical labeling.

Within the fixed token length, the relative size of s_{out} indicates the preference of hubs with lower node indices. Comparing Figures 4(a) and 4(b) of varying s_{out} under $s = 48$, it can be observed that the accuracy tends to increase on the homophilous CITESEER. On the opposite, the performance on CHAMELEON decreases when introducing more out labels under heterophily, showing that our Algorithm 3 sampling design is effective in retrieving distinct graph information by adjusting the hyperparameters of s_{out} and s_{in}.

5.4 Ablation Study

Table 4 examines the respective effectiveness of the hierarchical modules in the HubGT network architecture, where we separately present results on homophilous and heterophilous datasets. It can be observed that the model without SPD bias suffers the greatest accuracy drop, since topological information represented by positional encoding is necessary for GTs to retrieve the relative connection between nodes and gain performance improvement over learning plain node-level features.

In HubGT, the learnable virtual node representation is invoked to provide adaptive graph-level context before Transformer layers, while the attention-based node-wise readout module aims to distinguish nodes inside subgraphs and aggregate useful representation after encoder transformation. As shown in Table 4, both modules achieve relatively higher accuracy improvements on the heterophilous graph CHAMELEON, which validates that the proposed designs are particularly suitable for addressing the heterophily issue by recognizing hierarchical information.

Table 4: Ablation study of HubGT model components. The first line shows the accuracy of the complete HubGT architecture. Each subsequent line indicates the performance difference when the specified module is removed.

Dataset	CITESEER	Δ	CHAMELEON	Δ
HubGT	75.47	–	43.63	–
– Node Readout	72.21	-3.26	38.76	-4.87
– Virtual Node	71.15	-4.32	37.08	-6.55
– SPD Bias	68.55	-6.92	36.52	-7.11

6 CONCLUSION

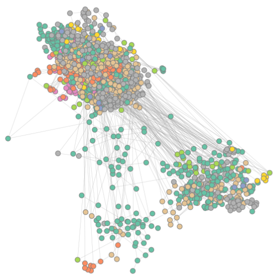
In this work, we present HubGT for leveraging decoupled graph hierarchy by hub labeling. Our analysis reveals that the label graph exhibits an informative hierarchy and enhances attention learning on the interaction between nodes. Regarding efficiency, construction and distance query of the label graph can be accomplished with *linear* complexity and are decoupled from iterative model training. Hence, the model benefits from scalability in computation speed and mini-batch training. Empirical evaluation showcases the superiority of HubGT, including efficacy under both homophily and heterophily, as well as efficient computation especially for inference on large-scale graphs of up to millions of nodes.

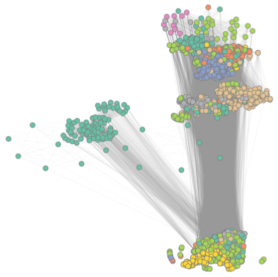
Table 5: Effectiveness and efficiency results on homophilous datasets. "Pre." , "Epoch", and "Infer" are precomputation, training epoch, and inference time (in seconds), respectively. "Mem." refers to peak GPU memory throughout the whole learning process (GB). Respective results of the first and second best performances on each dataset are marked in bold and underlined fonts.

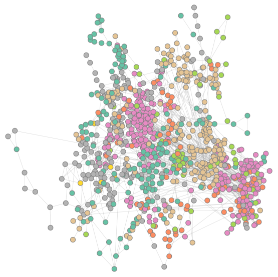
Homophilous	CORA					CITESEER					PUBMED				
	Pre.	Epoch	Infer	Mem.	Acc	Pre.	Epoch	Infer	Mem.	Acc	Pre.	Epoch	Infer	Mem.	Acc
DIFFormer*	-	0.11	0.13	1.2	83.37±0.50	-	0.07	0.07	1.7	<u>74.65</u>±0.67	-	0.37	0.35	2.7	75.77±0.40
PolyNormer*	-	0.11	0.65	1.4	80.43±1.55	-	0.21	0.86	1.6	68.70±0.95	-	0.86	6.07	2.5	75.80±0.46
NAGphormer	0.68	0.01	0.06	0.5	76.96±0.73	1.26	0.01	0.38	0.5	62.26±2.10	3.05	0.01	0.04	0.5	78.46±1.01
ANS-GT	43	2.0	1.12	2.0	<u>85.42</u>±0.52	59.9	11.65	4.25	11.9	73.58±0.98	529	14	3.52	1.9	<u>89.53</u>±0.51
GOAT	10.1	0.25	0.93	2.5	78.26±0.17	11.1	0.31	1.04	2.1	64.69±0.43	57.4	0.34	1.61	5.3	77.76±0.97
HSGT*	0.1	1.81	2.33	0.5	81.73±1.95	0.06	0.87	1.23	0.9	69.72±1.02	5.0	3.89	4.44	24	88.86±0.46
HubGT (ours)	0.42	0.20	0.02	2.5	**85.58**±0.18	0.43	0.24	0.02	2.0	**75.47**±2.22	2.6	1.47	0.16	1.7	**89.80**±0.48

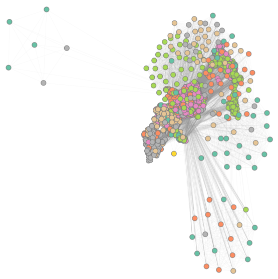
Heterophilous	CHAMELEON					SQUIRREL					TOLOKERS				
	Pre.	Epoch	Infer	Mem.	Acc	Pre.	Epoch	Infer	Mem.	Acc	Pre.	Epoch	Infer	Mem.	ROC AUC
DIFFormer*	-	0.09	0.38	0.50	37.83±4.54	-	0.05	0.05	0.7	35.73±1.37	-	0.16	85.8	0.88	74.88±0.59
PolyNormer*	-	0.03	0.17	1.1	40.70±3.38	-	0.07	0.49	1.2	**38.40**±1.10	-	1.27	15.5	9.4	79.39±0.50
NAGphormer	0.27	0.03	0.03	0.5	33.18±4.30	0.85	0.08	0.08	0.5	32.02±3.93	1.59	0.11	0.02	0.5	79.32±0.39
ANS-GT	11.2	1.98	0.78	2.8	<u>41.19</u>±0.69	28.1	4.48	1.95	6.6	37.15±1.10	716	2.37	3.42	10.7	79.31±0.97
GOAT	1.99	0.34	0.44	0.4	35.02±1.15	6.66	0.37	0.58	0.6	30.78±0.91	36.1	5.49	5.87	5.0	<u>79.46</u>±0.57
HSGT*	0.01	0.34	0.73	0.3	32.28±2.43	0.01	0.42	0.74	0.4	34.32±0.51	2.62	7.76	8.12	17.4	79.24±0.83
HubGT (ours)	0.04	0.08	0.007	1.6	**43.63**±2.34	0.24	0.18	0.01	2.6	<u>37.16</u>±0.57	1.4	0.99	0.02	2.2	**79.86**±0.47

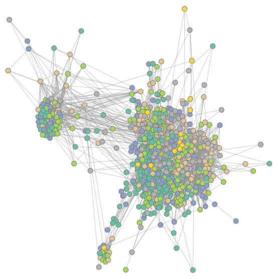
* Inference of these models is performed on the CPU in a full-batch manner due to their requirement of the whole graph.

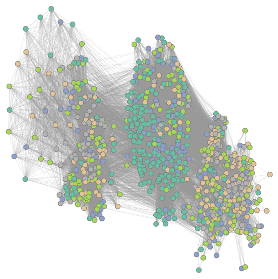

(a) CORA Raw Graph

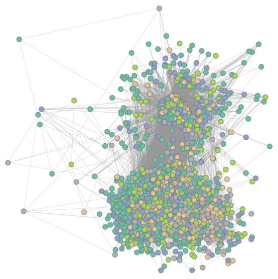

(b) CORA Label Graph


(c) CITESEER Raw Graph


(d) CITESEER Label Graph


(e) CHAMELEON Raw Graph


(f) CHAMELEON Label Graph


(g) SQUIRREL Raw Graph

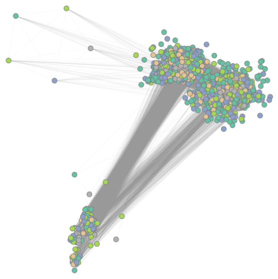

(h) SQUIRREL Label Graph

Figure 5: Visualization of the hierarchy of original and label graphs on realistic datasets. Color of each node denotes its class.

REFERENCES

[1] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. 2011. A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks. In *Experimental Algorithms*, Panos M. Pardalos and Steffen Rebennack (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 230–241.

[2] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. 2012. Hierarchical Hub Labelings for Shortest Paths. In *Algorithms – ESA 2012*, Leah Epstein and Paolo Ferragina (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 24–35.

[3] Takuya Akiba, Takanori Hayashi, Nozomi Nori, Yoichi Iwata, and Yuichi Yoshida. 2015. Efficient top-k shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[4] Takuya Akiba, Yoichi Iwata, Ken ichi Kawarabayashi, and Yuki Kawata. [n.d.]. Fast Shortest-path Distance Queries on Road Networks by Pruned Highway Labeling. *2014 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)* ([n. d.]), 147–154.

[5] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 349–360. https://doi.org/10.1145/2463676.2465315

[6] Wendong Bi, Lun Du, Qiang Fu, Yanlin Wang, Shi Han, and Dongmei Zhang. 2024. Make Heterophilic Graphs Better Fit GNN: A Graph Rewiring Approach. *IEEE Transactions on Knowledge and Data Engineering* 36, 12 (Dec. 2024), 8744–8757. https://doi.org/10.1109/TKDE.2024.3441766

[7] Chen Cai, Dingkang Wang, and Yusu Wang. 2021. Graph Coarsening with Neural Networks. In *9th International Conference on Learning Representations*.

[8] Cong Chen, Chaofan Tao, and Ngai Wong. 2021. Litegt: Efficient and lightweight graph transformers. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 161–170.

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

[10] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. 2023. NAGphormer: A Tokenized Graph Transformer for Node Classification in Large Graphs. In *11th International Conference on Learning Representations*.

[11] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks Via Importance Sampling. In *6th International Conference on Learning Representations*. 1–15.

[12] Ming Chen, Zhewei Wei, Bolin Ding, Yaliang Li, Ye Yuan, Xiaoyong Du, and Ji Rong Wen. 2020. Scalable graph neural networks via bidirectional propagation. *33rd Advances in Neural Information Processing Systems* (2020).

[13] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. 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*. 257–266.

[14] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. 2003. Reachability and Distance Queries via 2-Hop Labels. *SIAM J. Comput.* 32, 5 (2003), 1338–1355. https://doi.org/10.1137/S0097539702403098

[15] Chenhui Deng, Zichao Yue, and Zhiru Zhang. 2024. Polynormer: Polynomial-Expressive Graph Transformer in Linear Time. *The Twelfth International Conference on Learning Representations* (2024).

[16] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. 2020. GraphZoom: A multi-level spectral approach for accurate and scalable graph embedding. In *8th International Conference on Learning Representations*.

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

[18] Wenzheng Feng, Yuxiao Dong, Tinglin Huang, Ziqi Yin, Xu Cheng, Evgeny Kharlamov, and Jie Tang. 2022. GRAND+: Scalable Graph Random Neural Networks. In *Proceedings of the ACM Web Conference 2022*. ACM, 3248–3258. arXiv:2203.06389

[19] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, Jure Leskovec, Regina Barzilay, Peter Battaglia, Yoshua Bengio, Michael Bronstein, Stephan Günnemann, Will Hamilton, Tommi Jaakkola, Stefanie Jegelka, Maximilian Nickel, Chris Re, Le Song, Jian Tang, Max Welling, and Rich Zemel. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *33rd Advances in Neural Information Processing Systems* (2020).

[20] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. 2021. Scaling Up Graph Neural Networks Via Graph Coarsening. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Vol. 1. ACM, Virtual Event Singapore, 675–684.

[21] Md Shamim Hussain, Mohammed J Zaki, and Dharmashankar Subramanian. 2022. Global self-attention as a replacement for graph convolution. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 655–665.

[22] Pak Lon Ip, Shenghui Zhang, Xuekai Wei, Tsz Nam Chan, and Leong Hou U. 2024. Bridging Indexing Structure and Graph Learning: Expressive and Scalable Graph Neural Network via Core-Fringe. *OpenReview preprint* (2024). https://openreview.net/forum?id=j56A1HUTQS

[23] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[24] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized PageRank. *7th International Conference on Learning Representations* (2019), 1–15.

[25] Kezhi Kong, Jiuhai Chen, John Kirchenbauer, Renkun Ni, C Bayan Bruss, and Tom Goldstein. 2023. GOAT: A global transformer on large-scale graphs. In *International Conference on Machine Learning*. PMLR, 17375–17390.

[26] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. 2021. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems* 34 (2021), 21618–21629.

[27] Qimai Li, Xiaotong Zhang, Han Liu, Quanyu Dai, and Xiao-Ming Wu. 2021. Dimensionwise Separable 2-D Graph Convolution for Unsupervised and Semi-Supervised Learning on Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. ACM, Virtual Event Singapore, 953–963. https://doi.org/10.1145/3447548.3467413

[28] Ye Li, Leong Hou U, Man Lung Yiu, and Ngai Meng Kou. 2017. An experimental study on hub labeling based shortest path algorithms. *Proc. VLDB Endow.* 11, 4 (Dec. 2017), 445–457.

[29] Ningyi Liao, Siqiang Luo, Xiang Li, and Jieming Shi. 2023. LD2: Scalable Heterophilous Graph Neural Network with Decoupled Embedding. In *36th Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., 10197–10209.

[30] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. 2022. SCARA: Scalable Graph Neural Networks with Feature-Oriented Optimization. *Proceedings of the VLDB Endowment* 15, 11 (2022), 3240–3248. https://doi.org/10.14778/3551793.3551866 arXiv:2207.09179

[31] Ningyi Liao, Zihao Yu, and Siqiang Luo. 2024. HubGT: Scalable Graph Transformer with Decoupled Hierarchy Labeling [Tech Report]. https://github.com/nyLiao/HubGT/blob/main/Appendix.pdf

[32] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. 2021. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. In *34th Advances in Neural Information Processing Systems*.

[33] Jiahong Ma, Mingguo He, and Zhewei Wei. 2023. PolyFormer: Scalable Graph Transformer via Polynomial Attention. (2023).

[34] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. 2022. Is Homophily a Necessity for Graph Neural Networks?. In *10th International Conference on Learning Representations*. arXiv:2106.06134 [cs, stat]

[35] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. 2023. A Critical Look at Evaluation of GNNs under Heterophily: Are We Really Making Progress?. In *11th International Conference on Learning Representations*.

[36] Dian Ouyang, Dong Wen, Lu Qin, Lijun Chang, Xuemin Lin, and Ying Zhang. 2023. When hierarchy meets 2-hop-labeling: efficient shortest distance and path queries on road networks. *The VLDB Journal* 32, 6 (Nov. 2023), 1263–1287. https://doi.org/10.1007/s00778-023-00789-x

[37] Wonpyo Park, Woong-Gi Chang, Donggeon Lee, Juntae Kim, et al. 2022. GRPE: Relative Positional Encoding for Graph Transformer. In *ICLR2022 Machine Learning for Drug Discovery*.

[38] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. 2022. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems* 35 (2022), 14501–14515.

[39] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* 29, 3 (Sep. 2008), 93.

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

[41] Yuxuan Shi, Gong Cheng, and Evgeny Kharlamov. 2020. Keyword Search over Knowledge Graphs via Static and Dynamic Hub Labelings. In *Proceedings of The Web Conference 2020*. ACM, Taipei Taiwan, 235–245. https://doi.org/10.1145/3366423.3380110

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc.

[43] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. In *8th International Conference on Learning Representations*.

[44] Xiyuan Wang and Muhan Zhang. 2022. How Powerful are Spectral Graph Neural Networks. In *39th International Conference on Machine Learning*. arXiv:2205.11172 [cs]

[45] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *Proceedings of the 36th International Conference on Machine Learning*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. 6861–6871.

[46] Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. 2023. DIFFormer: Scalable (Graph) Transformers Induced by Energy Constrained Diffusion. In *The Eleventh International Conference on Learning Representations*.

[47] Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. 2022. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems* 35 (2022), 27387–27401.

[48] Qitian Wu, Wentao Zhao, Chenxiao Yang, Hengrui Zhang, Fan Nie, Haitian Jiang, Yatao Bian, and Junchi Yan. 2023. Simplifying and empowering transformers for large-graph representations. *Advances in Neural Information Processing Systems* 36 (2023).

[49] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. 2021. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems* 34 (2021), 13266–13279.

[50] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 2013. Fast and scalable reachability queries on graphs by pruned labeling with landmarks and paths. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 1601–1606.

[51] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *Advances in neural information processing systems* 34 (2021), 28877–28888.

[52] Zaixi Zhang, Qi Liu, Qingyong Hu, and Chee-Kong Lee. 2022. Hierarchical graph transformer with adaptive node sampling. *Advances in Neural Information Processing Systems* 35 (2022), 21171–21183.

[53] Haiteng Zhao, Shuming Ma, Dongdong Zhang, Zhi-Hong Deng, and Furu Wei. 2023. Are more layers beneficial to graph transformers? *The Eleventh International Conference on Learning Representations* (2023).

[54] Wenhao Zhu, Tianyu Wen, Guojie Song, Xiaojun Ma, and Liang Wang. 2023. Hierarchical transformer for scalable graph learning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. 4702–4710.

[55] Wenhao Zhu, Tianyu Wen, Guojie Song, Liang Wang, and Bo Zheng. 2023. On Structural Expressive Power of Graph Transformers. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. ACM, Long Beach CA USA, 3628–3637. https://doi.org/10.1145/3580305.3599451

[56] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks. In *33rd Advances in Neural Information Processing Systems*.