HubGT: Fast Graph Transformer with Decoupled Hierarchy Labeling

Ningyi Liao[*]
Nanyang Technological University
`liao0090@e.ntu.edu.sg`

Zihao Yu[*]
Nanyang Technological University
`zihao.yu@ntu.edu.sg`

Siqiang Luo[†]
Nanyang Technological University
`siqiang.luo@ntu.edu.sg`

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

Abstract

Graph Transformer (GT) leveraging the powerful Transformer architecture to learn graph-structured data. However, effectively representing graph information while ensuring efficiency remains challenging, as our analysis reveals that graph-scale operations still constitute the computational bottleneck in current GT designs and limit their applications to large graphs. In this work, we tackle the GT scalability issue by proposing HubGT, which is boosted by decoupled graph computation and hierarchical graph representations. HubGT represents graph information with a novel hub labeling scheme, which encompasses enriched neighborhoods for node token generation, and fast computation for distance-based positional encoding. Notably, the precomputation and training 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 offers efficient computation and mini-batch capability over existing GT designs on large-scale datasets while achieving top-tier effectiveness. Our code is available at: `https://github.com/gdmnl/HubGT`.

1 Introduction

Graph Transformers (GTs) has recently emerged as a family of neural networks that introduce the advantageous Transformer architecture [1] 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 [2, 3, 4, 5]. 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 apply Graph Transformers to large graph data, 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 [6, 7, 8], 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 [9, 10], polynomial spectral transformation [11, 12], and

[*]Both authors contributed equally to this research.
[†]Corresponding author.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

Table 1: Complexity analysis on GT models. "**Precomputation**" and "**training time**" represent the complexity of one-time processing and iterative forward-passing computation, respectively. "**RAM**" and "**GPU memory**" indicate the sizes of overall data and variable representations during learning. Data "**scale**" is represented by the largest graph node size n and edge size m used in the original papers.

Taxonomy	Model	Precompute Time	Train Time	RAM Mem.	GPU Mem.	Scale
Vanilla (FB)	Graphormer [2]	$O(n^3)$	$O(Ln^2F)$	$O(n^2)$	$O(Ln^2F)$	0.3K/0.6K
	GRPE [18]	$O(n^2)$	$O(Ln^2F)$	$O(n^2)$	$O(Ln^2F)$	0.3K/0.6K
Kernel-based (NS)	GraphGPS [7]	$O(n^3)$	$O(LnF^2 + LmF)$	$O(nF + n^2)$	$O(Ln_bF + m)$	1.0K/3.0K
	Exphormer [19]	$O(dn)$	$O(LnF^2 + LmF)$	$O(nF + m)$	$O(Ln_bF + m)$	0.2M/1.2M
	NodeFormer [6]	–	$O(LnF^2 + LmF)$	$O(nF + m)$	$O(Ln_bF + m)$	2.4M/60M
	DIFFormer [8]	–	$O(LnF^2 + LmF)$	$O(nF + m)$	$O(Ln_bF + m)$	1.6M/40M
	PolyNormer [12]	–	$O(LnF^2 + LmF)$	$O(nF + m)$	$O(Ln_bF + m)$	2.4M/0.1B
Hierarchical (RS)	NAGphormer [9]	$O(LmF_0)$	$O(LnF^2)$	$O(LnF)$	$O(Ln_bF^2)$	2.4M/60M
	PolyFormer [11]	$O(LmF_0)$	$O(LnF^2)$	$O(LnF)$	$O(Ln_bF^2)$	0.2M/30M
	ANS-GT [13]	$O(ns^2 + md^L)$	$O(LnF^2 + Lns^2F + Lm)$	$O(nF + ns^2 + md^L)$	$O(Ln_bF + n_bs^2)$	20K/0.2M
	GOAT [10]	$O(nF)$	$O(LnF^2 + LmF)$	$O(nF + m)$	$O(Ln_b^2 + Ln_bF + m)$	2.9M/0.1B
	HSGT [14]	$O(n + md^L)$	$O(LnF^2 + LmF)$	$O(nF + md^L)$	$O(Ln_b^2 + Ln_bF + Lm)$	2.4M/0.1B
	HubGT (ours)	$O(ns^3 + ms)$	$O(LnF^2 + Lns^2F)$	$O(nF + ns^2)$	$O(Ln_bF + n_bs^2)$	1.6M/0.1B

Notations: L and F are model depth and width, respectively. $d = m/n$ is the average degree of the graph. s is the sample size for subgraph-based methods.

hierarchical graph coarsening [13, 14]. However, we identify two major drawbacks of existing scalable GTs: in terms of efficacy, most models primarily concentrate on local topology, which is highly diluted in large graphs and undermines GT expressivity in capturing global information; in terms of efficiency, graph-scale operations still persist throughout their training iterations, and the computational overhead substantially increases as the graphs become larger.

In this work, we specifically target the above two challenges brought by scalability. 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 [15, 16, 17]. We innovatively introduce the graph label hierarchy to enhance GT capability by establishing global connections to influential graph hubs, which is shown to provide a more inclusive receptive field than the conventional adjacency-based scheme. 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.

Then, to address the efficiency issue, we design a three-level index for computing and storing the graph data, where the dedicated computation process and acquired graph labels share the same hierarchy, comprising of both local edges and global hubs. The indexing process can be fully decoupled as precomputation with $O(m)$ overhead from the iterative training. Then, querying SPD on the index can be performed in only $O(1)$ 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 our graph representation based on hub labeling. We summarize the contributions of this work as follows:

• We propose HubGT as an efficient 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 fast and 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

Graph Transformer and Positional Encoding. Consider a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. The graph adjacency matrix is $\boldsymbol{A}$ and average degree is $d = m/n$. 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 [1] 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:

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

where $\cdot\|\cdot$ denotes the matrix concatenation operation. The projection dimension is usually set as $F_K = F_V = F/N_H$. Beside the representation $\boldsymbol{H}$, positional encoding (PE) $\boldsymbol{P}$ in Eq. (2) can also incorporate graph topology into the GT attention module by encoding pair-wise information. Typical encoding approaches include graph proximity [2, 13, 20], Laplacian eigenvectors [21, 22, 4], and shortest path distance [18, 23, 24].

Scalable GT and Training Schemes. The majority of vanilla GTs [21, 2, 3] are typically proposed for graph-level learning tasks on small graphs with full-batch training (FB). This is relevant to their quadratic complexity as revealed by Table 1. For large-scale graphs, model scalability is primarily dominated by the graph-scale terms m and n, which renders a critical bottleneck in these models as indicated in Eq. (2), due to representing n nodes with $O(n^2 F)$ time and memory overhead.

To mitigate the quadratic complexity and foster scalable learning, scalable GTs employ mini-batch training, which replaces the full representation $\boldsymbol{H}$ with batches containing n_b nodes and reduces memory complexity to $O(n_b^2 F)$. *Kernel-based GTs* [6, 7, 8, 19, 12] 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.

Alternatively, *hierarchical GTs* [9, 11, 10, 13, 14] exploit the power of GTs to learn node relations by embedding node-level identity through the input data $\boldsymbol{X}$. Its core design is crafting an effective embedding scheme to comply with GT expressivity. Since the graph topology is embedded in an permutation-invariant manner, mini-batching can be performed through random sampling (RS). The model can enjoy better scalability if the graph processing is fully independent of GT attention. 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 [9] and PolyFormer [11], still involve graph-level operations during training as in Table 1, which hinders GPU utilization and causes additional overhead. A thorough analysis of the related models can be found in Appendix B.

3 Motivating Study

Motivation 1: Graph hierarchy beyond adjacency. The expressiveness of GTs mainly stems from the full-graph attention formulated in Eq. (2), which captures critical node pairs in the graph topology to learn node representations [2, 6]. However, since the mini-batch scheme replaces it with in-batch attention, its capability is potentially hindered. To compensate the information loss, scalable GTs usually invoke more powerful embedding and encoding techniques, enhancing the *global* graph view for more candidate nodes by expanding the receptive field. In canonical mini-batch GT models [2, 6, 7, 9], the graph information is typically derived from the graph *adjacency* $\boldsymbol{A}$, which symbolizes neighborhood information $\mathcal{E}$. For both kernel-based and hierarchical GT variants, their expressiveness in distinguishing different graph structures is characterized by the substructure used in attention tokens [24].

However, recent advances in message-passing GNNs reveal that, the adjacency alone is insufficient for retrieving topological information in graph learning [25]. In complicated scenarios such as heterophilous graphs, the local graph topology may be ambiguous or even misleading [26, 27, 28]. To illustrate, Figure 1 shows the distribution of node neighbors considering their classification labels depicted by the homophily score [29]. A lower homophily score implies high heterophily, where less

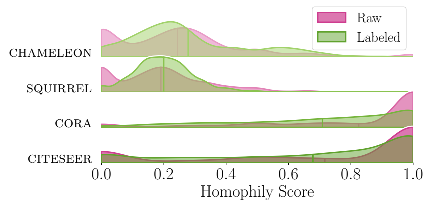
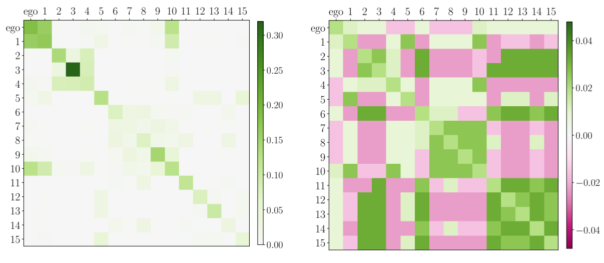


Figure 1: Relative distribution of homophily scores between original and extended graphs. A higher score implies more similar nodes in the neighborhood.



(a) L-hop Adjacency (b) SPD

Figure 2: Comparison of PEs as attention biases on CITESEER.

neighbors share the same label with the ego node. On graphs such as CHAMELEON and SQUIRREL, a large portion of nodes exhibits zero homophily, indicating that substructures depending on graph edges $\mathcal{E}$ barely include neighbors with the same label for GT to attend.

We are thence motivated to improve GT by augmenting the existing graph connections $\mathcal{E}$ to an extended edge set $\hat{\mathcal{E}}$ containing additional node interactions beyond the neighborhood. As shown in Figure 1, establishing more edges enhances nodes of lower scores with more homophilous connections, effectively addressing the zero-homophily issue. Therefore, encompassing both local connections and global knowledge is capable of forming a hierarchical structure beyond the original graph adjacency, which benefits the model in retrieving a wide spectrum of information [13, 10, 14].

Motivation 2: Dense PEs for large graphs. While an array of PE schemes are utilized for explicitly integrating graph knowledge into GT learning, they are shown to possess various information aspects and expressiveness [2, 7, 5]. In particular, shortest path distance (SPD) as a form of *pairwise* PE [2] is revealed to be at least expressive as the Weisfeiler-Lehman (WL) graph isomorphism test [5] and empirically superior in graph classification tasks [30, 7, 31]. However, the conventional Floyd-Warshall algorithm for calculating whole-graph SPD entails $O(n^3)$ time and $O(n^2)$ space, which is prohibitive to large graphs.

On the other hand, scalable GTs commonly employ *sparse* PEs such as L-hop Adjacency $(\boldsymbol{D}^{-1/2}(\boldsymbol{A}+\boldsymbol{I})\boldsymbol{D}^{-1/2})^L$, which is straightforward to acquire by sparse matrix multiplication in $O(md^L)$ complexity [13, 14]. However, since nodes in large graphs are loosely connected, the in-batch PE under mini-batch training can be too sparse to represent pairwise relationships. As observed in Figure 2(a), PE values are only significant between the ego node and two direct neighbors, while diminishing to zero for more distant nodes in the same batch. The values for non-ego node pairs are also small, rendering the bias ineffective in attending to node-pair interactions.

To this end, we attempt to apply the more effective pairwise SPD for encoding arbitrary node pairs on large graphs. Figure 2(b) suggests that the scheme is superior in providing rich information covering both positive and negative relationships, even for nodes that are not directly connected. Therefore, it is also more suitable for the extended hierarchical graph structure $\hat{\mathcal{E}}$ involving global connections to distant nodes.

4 Efficient Hierarchical Labeling

In light of the limitations of existing scalable GTs on large graphs, our HubGT aims to address these two challenges simultaneously while ensuring computational efficiency: (1) The dedicated data structure provides expressive graph information for generating node *embeddings* that represent the graph hierarchy beyond mere adjacency. (2) Node-pair SPD can be efficiently calculated based on the hierarchy, which is then used as positional *encoding*. In this section, we first introduce the concept and novel settings of our graph labeling approach. Then, we focus on designing the respective indexing and querying algorithms, while their integration into GT learning is elaborated in Section 5.

4.1 Problem Statement

Conventional Hub Labeling: Full-graph SPD. Computing the shortest path between graph nodes is a fundamental problem in graph analysis. Canonical hub labeling algorithms [32, 33, 34] are proposed to efficiently answer full-graph SPD queries between arbitrary node pairs. To achieve this, an index $\mathcal{L}(v)$ is constructed for each node $v \in \mathcal{V}$, containing a number of labels. Each *label* for node v typically consists of an end node u and the corresponding shortest distance $b(u,v)$ between the node pair. Then, given a query involving two arbitrary nodes $u, v \in \mathcal{V}$, the SPD can be promptly answered by looking up and combining relevant labels from the graph index $\mathcal{L}$, thereby preventing exhaustive traversal of the entire graph.

Our Settings: Subgraph SPD. In HubGT, we construct the graph index through a separate *precomputation* phase that occurs prior to GT learning. Note that each label represents a node pair (u,v) and the distance $b(u,v)$. Hence, we are able to extend the graph structure to a weighted, directed graph by regarding the labels as graph edges, i.e., $\hat{\mathcal{G}} = \langle \mathcal{V}, \hat{\mathcal{E}} \rangle$, where $\hat{\mathcal{E}} = \{(u,v) \mid u \in \mathcal{L}(v)\}$. Appendix A.1 provides a thorough elaboration on the theoretical properties of the extended graph when generated by our hub labeling approach.

To facilitate node representations for GT learning, here, we consider an SPD querying problem different from conventional full-graph queries, which allows HubGT to simultaneously perform node token generation and SPD calculation in an end-to-end manner. The subgraph SPD problem can be explained as follows: Given a node r, we aim to (1) sample a subgraph, i.e., a set of nodes $\mathcal{S}(r)$, such that $(u,r) \in \hat{\mathcal{E}}$ or $(r,u) \in \hat{\mathcal{E}}$ for $u \in \mathcal{S}(r)$ solely based on the built index; (2) at the same time, acquire SPDs $b(u,v)$ for all node pairs $u, v \in \mathcal{S}(r)$. Considering the iterative nature of such queries during training, our primary design objective is to minimize querying overhead while maintaining reasonable time and space indexing overhead.

4.2 HubGT Indexing

Under the conventional full-graph SPD problem, the representative Pruned Landmark Labeling (PLL) algorithm [35, 36] offers an indexing strategy with efficient search space and minimal index size. The algorithm searches labels by a pruned Breadth First Search (BFS) for each node following a particular order. During construction, PLL tends to regard foremost nodes in search as *hubs* and connecting more edges to them, while eliminating the labeling from latter nodes. Hence, a hierarchy of nodes are intrinsically formed by labeling.

To address the specific requirements of node sampling and SPD calculation in the subgraph SPD problem, we design the HubGT algorithm for hub labeling, which is based on the intuition of search-and-prune from PLL. This leads to a distinct three-level hierarchy in both index construction and querying, with the aim to strategically balance index size and query time. Our Algorithms 1 to 3 are detailed in Appendix A.2

Hierarchy-1 (H-1): Local hub labeling. Unlike the SPD query on an arbitrary node pair addressed by PLL, the query for $b(u,v)$ in HubGT always orients an intermediary node r. In other words, u, v are at most 2-hop neighbors in $\hat{\mathcal{E}}$. To leverage this relevance, we broaden the derivation of 2-hop labeling in [35] to these subgraph nodes, that the distance candidate orienting $b_r(u,v)$ can be acquired by utilizing the connectivity with the local hub node 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 $\mathcal{M}_r^i(v) = \{w \in \mathcal{N}(r) \,|\, b(r,v) - b(w,v) = i\}$, $i = -1, 0,$ 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. On top of the pruned BFS in typical PLL, we also 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 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 label in $\mathcal{L}(v)$ or $\mathcal{L}'(v)$ is a quadruple $(u, b(u,v), \mathcal{M}_v^1(u), \mathcal{M}_v^0(u))$, which suffices the distance query based on Eq. (3). In specific, we explicitly impose a maximum capacity $O(s)$ for each node label $\mathcal{L}(v)$ in Algorithm 1. This is because at most s neighbors are required to form the subgraph in our problem.

Hierarchy-2 (H-2): Global hub labeling. H-1 index presents the idea of utilizing the relative position orienting a given hub r to compute the distances of 2-hop node pairs in a query. It can be further generalized to some *global* hubs by indexing their labels as another level of hierarchy, which corresponds to the bit-parallel BFS in [35]. More specifically, we select a small set of nodes with low indices in the search order, and perform Algorithm 1 BFS without pruning. 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 candidate distance of an arbitrary node pair $b_t^{(2)}(u,v)$ can be similarly computed by Eq. (3) orienting hub t.

Hierarchy-0 (H-0): 2-hop pair caching. Although the H-1 and H-2 query Eq. (3) produces the shortest distance $b(u,v)$ when $b_r^{(1)}(u,v) \leq b(u,v) + \delta_r(u,v)$, it is possible that the actual shortest path $\mathcal{P}(u,v)$ does not pass through r or its neighbors. In this case, the exact SPD calculation 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 as it guarantees answer to any queries. However, calculation by Eq. (4) results in a complexity of $O(|\mathcal{L}(u)| + |\mathcal{L}(v)|)$, which is still not satisfying under the repetitive querying 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 $I_v[u]$ for $u < v$. By employing an appropriate 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 $O(1)$.

The precomputation for building the H-0 index is 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.

4.3 HubGT Querying

After building the indices H-2, H-1, and H-0 successively, querying label neighbors $\mathcal{S}(r)$ and their distances as in Section 4.1 can be achieved solely on the $\mathcal{L}, \mathcal{L}', \mathcal{I}$ without resorting to the raw 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. 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.

As analyzed in Appendix A.1, generating subgraph structures from the constructed labels preserves local neighbors while adding global hubs (Property 1). This is preferable than the traditional adjacency-based neighborhood sampling 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 (Property 2). Meanwhile, non-hub nodes, usually the less significant nodes under heterophily, are constrained in local substructures with more similar neighbors (Property 3). From the efficiency aspect, it maintains the form of 1-hop sampling and prevents the exponential multi-hop operations.

Complexity Analysis. Let the label size $|\mathcal{L}(v)|$ bounded by $O(s)$, H-1 and H-2 indexing complete the traversal of all nodes in $O(ns + ms)$ time. 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 highlight that the empirical label sizes observed in experiments 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

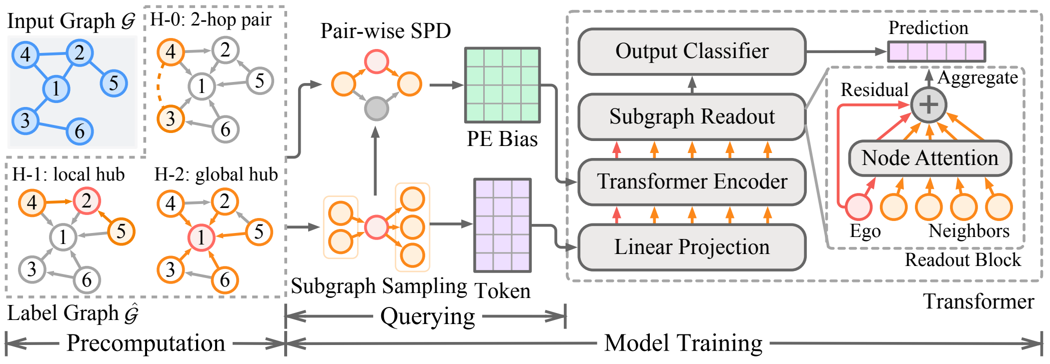


Figure 3: HubGT framework with precomputation and training stages. The **precomputation** stage processes the input graph and constructs the label index. During **training**, subgraph nodes and SPDs are queried from the index and applied as different Transformer inputs. Querying on CPU and training on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

each training iteration, with each query node possessing at most s^2 pair-wise distance computations, entails $O(ns^2)$ overhead.

5 HubGT Framework

Once the index is built in precomputation, it remains static in the RAM throughout GT training iterations. To efficiently exploit the labeling structure in HubGT, in this section, we elaborate our end-to-end design querying graph hierarchy for both graph embeddings and positional encoding to facilitate GT learning. Figure 3 illustrates the overview of the pipeline.

5.1 Graph Information Querying

Subgraph for Node Embeddings. We leverage the subgraph hierarchy in the index, i.e., neighboring nodes in labels $\mathcal{L}(v)$ and $\mathcal{L}'(v)$ to generate node embeddings. Since the neighborhood size is variable, we convert it into a fixed-length token $\mathcal{S}(v)$ by sampling as in Algorithm 3. The relative values of hyperparameters s_{in} and s_{out} can be used to balance the ratio between local long-tailed nodes and distant landmark nodes in $\mathcal{G}$, respectively. The node embedding is generated by concatenating the raw attributes $\boldsymbol{X}[u]$ of subgraph nodes $u \in \mathcal{S}(v)$, denoted as $\boldsymbol{X}[\mathcal{S}(v)]$.

SPD for Positional Encoding. Thanks to the efficient graph labeling, 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 $\boldsymbol{P}[u, v] = f_B(\boldsymbol{B}_r[u, v])$.

Global Hubs. Section 4.2 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 [2] 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.

5.2 Representation Learning

Since our contribution focus on the labeling process, HubGT adapts the scalable GT architecture [2, 13] incorporating subgraph precomputation and mini-batch training, which can potentially be enhanced by further acceleration techniques developed for the Transformer architecture.

The input subgraph embeddings are first projected as $\boldsymbol{H}[v] = \text{MLP}_X(\boldsymbol{X}[\mathcal{S}(v)])$ by an MLP_X. 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 before the output classifier MLP_Z:

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

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.

Remarkably, querying a batch of nodes by Algorithm 3 can be conducted in parallel. The SPD query on CPU can be pipelined with GT training on GPU, so that training can be performed seamlessly without blocked by the querying process. 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, only strides of $\boldsymbol{X}$ and batches of $\boldsymbol{B}_r$ are loaded onto GPU.

6 Experiments

6.1 Experimental Settings

Tasks and Datasets. We focus on the node classification task on 14 benchmark datasets covering both homophily [37, 38, 39] and heterophily [40, 41]. 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 Appendix C. 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 Exphormer [19], DIFFormer [8], and PolyNormer [12] are considered as kernel-based approaches. NAGphormer [9], GOAT [10], HSGT [14], and ANS-GT [13] stand for hierarchical GTs. Three message-passing GNNs, including GCNJK[42], MixHop [43], and SGFormer [44] are also included for comparison.

6.2 Performance Comparison

Table 2 presents the efficacy and efficiency evaluation results on 8 large-scale graphs, while results on other datasets can be found in Appendix D.2. Visualization of the experiment results are presented in Figure 4. 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 on 11 out of 14 datasets, demonstrating a favorable balance between effectiveness and efficiency.

Time Efficiency. Benefiting from the decoupled architecture, HubGT is powerful in achieving competitive learning and inference speeds with existing efficiency-oriented GTs. It consistently showcases the fastest inference, since Section 5.2 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$\times$ 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.

Table 2: Effectiveness and efficiency results on large-scale graph datasets, while more evaluations are in Appendix D.2. "**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**
GCNJK	-	0.49	0.05	94.67±0.02	-	0.29	0.02	60.03±0.32	-	1.6	0.16	88.99±0.39	-	0.96	0.10	28.65±0.03
MixHop	-	1.0	0.07	95.37±0.12	-	0.79	0.05	60.38±0.08	-	1.6	0.12	91.61±0.13	-	0.68	0.06	30.52±0.04
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
Exphormer	84	1.7	1.0	96.08±0.11	32	1.8	0.46	67.87±0.18	(OOM)				(OOM)			
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
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**
GCNJK	-	0.35	0.04	65.91±0.16	-	0.06	0.03	80.65±0.07	-	0.15	0.03	59.91±0.42	-	0.06	0.09	59.38±0.21
MixHop	-	0.31	0.04	75.00±0.37	-	0.08	0.01	80.63±0.04	-	0.11	0.01	61.80±0.01	-	0.15	0.03	64.02±0.02
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
Exphormer	(OOM)				(OOM)				42	1.3	0.41	64.24±0.35	(OOM)			
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	-	1.5	89	64.72±0.65	-	4.2	67	**81.03**±0.08
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

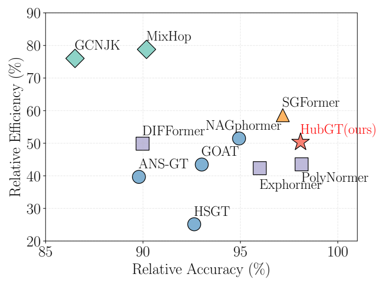
* Inference of these models is performed on the CPU in a full-batch manner due to their requirement of the whole graph.

We additionally note that while some models claim to be applicable to million-scale graphs as shown in Table 1, they exhibit excessive training time in our settings and fail to produce convergent results in one day. Hence, we only record the efficiency evaluations in Table 2. 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.

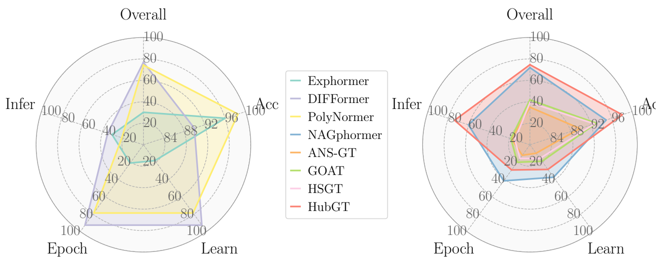
Memory Footprint. In modern computing platforms, GPU memory is usually highly constrained and becomes the scalability bottleneck for the resource-intensive graph learning. HubGT exhibits efficient utilization of GPU for training with larger batch sizes while avoiding the out-of-memory issue. In comparison, 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 Table 5, 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 4.1. 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



(a) Overall Efficiency vs Acc



(b) Metrics for kernel-based (left) and hierarchical (right) GTs

Figure 4: Relative model compassion for average evaluation metrics (in %) across 8 datasets in Table 2. **(a)** Trade-off between efficiency and effectiveness. Models on the upper right corner exhibit better efficiency and accuracy. HubGT showcases a favorable balance for achieving top-tier accuracy with better efficiency. **(b)** Detailed relative scores for respective metrics and models. Detailed computation for the scores are explained in Appendix D.2.

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 Appendix D.2.

6.3 Ablation Study

Table 3 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.

Table 3: 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	–
w/o Node Readout	72.21	-3.26	38.76	-4.87
w/o Global Hubs	71.15	-4.32	37.08	-6.55
w/o SPD Bias	68.55	-6.92	36.52	-7.11

In HubGT, the learnable global hub 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 3, 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.

7 Conclusion

In this work, we present HubGT, a novel PE calculation featuring the decoupled graph hierarchy through hub labeling. Our analysis reveals that the label graph exhibits an informative hierarchy and enhances GT 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.

Acknowledgments and Disclosure of Funding

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG3-RP-2024-034), Singapore MOE AcRF Tier-2 funding (MOE-T2EP20122-0003), and NTU startup grant (020948-00001). Ningyi Liao is supported by the Joint NTU-WeBank Research Centre on FinTech, Nanyang Technological University, Singapore. Zihao Yu is supported by A*STAR RIE2025 Manufacturing, Trade and Connectivity (MTC) Programmatic Fund (M24N6b0043) administered by A*STAR.

References

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

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

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

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

[5] Wenhao Zhu, Tianyu Wen, Guojie Song, Liang Wang, and Bo Zheng. On structural expressive power of graph transformers. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3628–3637, Long Beach CA USA, August 2023. ACM.

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

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

[8] Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. Difformer: Scalable (graph) transformers induced by energy constrained diffusion. In *The Eleventh International Conference on Learning Representations*, 2023.

[9] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. In *11th International Conference on Learning Representations*, February 2023.

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

[11] Jiahong Ma, Mingguo He, and Zhewei Wei. Polyformer: Scalable graph transformer via polynomial attention. 2023.

[12] Chenhui Deng, Zichao Yue, and Zhiru Zhang. Polynormer: Polynomial-expressive graph transformer in linear time. *The Twelfth International Conference on Learning Representations*, 2024.

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

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

[15] Yosuke Yano, Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 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*, pages 1601–1606, 2013.

[16] Takuya Akiba, Yoichi Iwata, Ken ichi Kawarabayashi, and Yuki Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. *2014 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 147–154.

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

[18] Wonpyo Park, Woong-Gi Chang, Donggeon Lee, Juntae Kim, et al. Grpe: Relative positional encoding for graph transformer. In *ICLR2022 Machine Learning for Drug Discovery*, 2022.

[19] Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J. Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. In *Proceedings of the 40th International Conference on Machine Learning*, pages 31613–31632. PMLR, July 2023.

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

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

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

[23] 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.

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

[25] Wendong Bi, Lun Du, Qiang Fu, Yanlin Wang, Shi Han, and Dongmei Zhang. Make heterophilic graphs better fit gnn: A graph rewiring approach. *IEEE Transactions on Knowledge and Data Engineering*, 36(12):8744–8757, December 2024.

[26] Qimai Li, Xiaotong Zhang, Han Liu, Quanyu Dai, and Xiao-Ming Wu. 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*, pages 953–963, Virtual Event Singapore, August 2021. ACM.

[27] Xiyuan Wang and Muhan Zhang. How powerful are spectral graph neural networks. In *39th International Conference on Machine Learning*, June 2022.

[28] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *10th International Conference on Learning Representations*, 2022.

[29] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *8th International Conference on Learning Representations*. arXiv, February 2020.

[30] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. GraphiT: Encoding graph structure in transformers, June 2021.

[31] Liheng Ma, Chen Lin, Derek Lim, Adriana Romero-Soriano, Puneet K. Dokania, Mark Coates, Philip Torr, and Ser-Nam Lim. Graph inductive biases in transformers without message passing. In *40th International Conference on Machine Learning*, volume 202, pages 23321–23337. PMLR, May 2023.

[32] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In Panos M. Pardalos and Steffen Rebennack, editors, *Experimental Algorithms*, pages 230–241, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[33] Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. Hierarchical hub labelings for shortest paths. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, pages 24–35, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[34] Dian Ouyang, Dong Wen, Lu Qin, Lijun Chang, Xuemin Lin, and Ying Zhang. When hierarchy meets 2-hop-labeling: efficient shortest distance and path queries on road networks. *The VLDB Journal*, 32(6):1263–1287, November 2023.

[35] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. 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*, pages 349–360. ACM, June 2013.

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

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

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

[39] 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. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *33rd Advances in Neural Information Processing Systems*, 2020.

[40] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *34th Advances in Neural Information Processing Systems*, 2021.

[41] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at evaluation of gnns under heterophily: Are we really making progress? In *11th International Conference on Learning Representations*, 2023.

[42] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *35th International Conference on Machine Learning*, volume 80. PMLR, 2018.

[43] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In *36th International Conference on Machine Learning*, volume 97. PMLR, 2019.

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

[45] Yuxuan Shi, Gong Cheng, and Evgeny Kharlamov. Keyword search over knowledge graphs via static and dynamic hub labelings. In *Proceedings of The Web Conference 2020*, pages 235–245, Taipei Taiwan, April 2020. ACM.

[46] Guoming Li, Jian Yang, Shangsong Liang, and Dongsheng Luo. Polynomial selection in spectral graph neural networks: An error-sum of function slices approach. In *Proceedings of the ACM Web Conference*, January 2025.

[47] Simon Markus Geisler, Arthur Kosmala, Daniel Herbst, and Stephan Günnemann. Spatio-spectral graph neural networks. *Advances in Neural Information Processing Systems*, 37:49022–49080, December 2024.

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

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

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

[51] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 6861–6871, 6 2019.

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

[53] Ningyi Liao, Dingheng Mo, Siqiang Luo, Xiang Li, and Pengcheng Yin. Scara: Scalable graph neural networks with feature-oriented optimization. *Proceedings of the VLDB Endowment*, 15(11):3240–3248, 2022.

[54] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations*, pages 1–15, 2018.

[55] 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.

[56] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. In *33rd Advances in Neural Information Processing Systems*, 2019.

[57] Wenzheng Feng, Yuxiao Dong, Tinglin Huang, Ziqi Yin, Xu Cheng, Evgeny Kharlamov, and Jie Tang. Grand+: Scalable graph random neural networks. In *Proceedings of the ACM Web Conference 2022*, pages 3248–3258. ACM, April 2022.

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

[59] Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, volume 1, pages 675–684, Virtual Event Singapore, August 2021. ACM.

[60] Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *9th International Conference on Learning Representations*, February 2021.

[61] Ningyi Liao, Siqiang Luo, Xiang Li, and Jieming Shi. Ld2: Scalable heterophilous graph neural network with decoupled embedding. In *36th Advances in Neural Information Processing Systems*, volume 36, pages 10197–10209. Curran Associates, Inc., December 2023.

[62] Pak Lon Ip, Shenghui Zhang, Xuekai Wei, Tsz Nam Chan, and Leong Hou U. Bridging indexing structure and graph learning: Expressive and scalable graph neural network via core-fringe. *OpenReview preprint*, 2024.

[63] Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. FlexAttention: A programming model for generating fused attention variants. In *8th Conference on Machine Learning and Systems*, May 2025.

[64] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *7th International Conference on Learning Representations, ICLR 2019 - Workshop Track Proceedings*, 2019.

[65] Yash Deshpande, Subhabrata Sen, Andrea Montanari, and Elchanan Mossel. Contextual stochastic block models. In *31st Advances in Neural Information Processing Systems*, 2018.

[66] Seiji Maekawa, Yuya Sasaki, George Fletcher, and Makoto Onizuka. GenCAT: Generating attributed graphs with controlled relationships between classes, attributes, and topology. *Information Systems*, 115:102195, 2023.

[67] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *Proceedings of the 23rd international conference on World wide web*, pages 237–248. ACM, April 2014.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: We accurately claim our contributions in Section 1.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations and potential improvements are discussed in Appendix E.

   Guidelines:

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

3. **Theory Assumptions and Proofs**

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

   Answer: [Yes]

Justification: Preliminary and assumptions are provided in Section 4.1 and Appendix A.

Guidelines:

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

4. **Experimental Result Reproducibility**

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

   Answer: [Yes]

   Justification: Experiment details and configurations are in Appendix C.

   Guidelines:

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

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

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

Answer: [Yes]

Justification: Reproducibility instructions and scripts are available in the codebase.

Guidelines:

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

6. **Experimental Setting/Details**

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

   Answer: [Yes]

   Justification: Experiment details are in Appendix C.

   Guidelines:

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

7. **Experiment Statistical Significance**

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

   Answer: [Yes]

   Justification: Error bars are reported in main experiments.

   Guidelines:

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

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

8. **Experiments Compute Resources**

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

   Answer: [Yes]

   Justification: Experiment details are in Section 6.1 and Appendix C.

   Guidelines:

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

9. **Code Of Ethics**

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

   Answer: [Yes]

   Justification: We conform the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader Impacts**

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

    Answer: [Yes]

    Justification: Broader impacts are discussed in Appendix E.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to

generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [NA]

    Justification: This paper is for foundational research and we do not foresee such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: Baselines and datasets are cited and referred in the codebase and discussed in Appendix B.

    Guidelines:

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

13. **New Assets**

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

Answer: [Yes]

Justification: Our model setups are in Appendix C. Tuning scripts are in the codebase.

Guidelines:

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

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

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

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

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

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

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

Answer: [NA]

Justification: The paper does not require IRB approvals.

Guidelines:

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

A Detailed Theoretical Analysis

A.1 Label Graph Properties

In SPD calculation, the index $\mathcal{L}(v)$ is said to form a *2-hop cover* if for an arbitrary node pair $u, v \in \mathcal{V}$, there exists a node $w \in \mathcal{L}(u) \cap \mathcal{L}(v)$. A 2-hop cover ensures that any SPD queries can be acquired by Eq. (4). A straight-forward approach to build the 2-hop cover is to traverse the whole graph for each node successively. This is, however, prohibitive due to the repetitive traversal and overlapped labels.

As briefed in Section 4.2, the Pruned Landmark Labeling (PLL) algorithm searches labels by a pruned BFS for each node in $\mathcal{V}$ relying on a particular order. Denoting each node by a unique index $1, \cdots, n$, $u < v$ indicates that node u precedes node v in the sequence. The labeling process performs a pruned Breadth First Search (BFS) for each node from 1 to n, adding hubs with shorter distances into the label set while eliminating the search from insignificant nodes. Notably, the algorithm is agnostic to the search order. In this work, we follow [15] 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 [45, 34], they nonetheless incur additional computational overhead.

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. 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 $\mathcal{E}$.

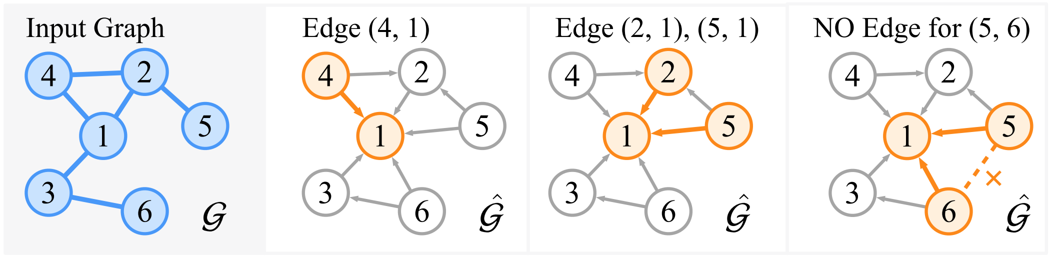
We summarize the following three properties of the label hierarchy produced by PLL:

Property 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 $u > v$.*

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 5(a) is represented by the directed edge $(4, 1)$ in Figure 5(b). Property 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 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 $w > v$.*

[35] 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 5(a), the shortest path between $(1, 5)$ passing node 2 results in edges $(2, 1)$ and $(5, 1)$ in Figure 5(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



(a) Raw Graph (b) Property 1 (c) Property 2 (d) Property 3

Figure 5: 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.

appear in shortest paths more frequently, and consequently link to a majority of nodes, including those long-tailed low-degree nodes in $\hat{\mathcal{G}}$.

Property 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 $(u, v) \notin \hat{\mathcal{E}}$.*

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 5(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.

In brief, Property 1 ensures that neighboring nodes in the raw graph $\mathcal{G}$ are still connected in $\hat{\mathcal{G}}$. Properties 2 and 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.

A.2 Discussion on Indexing

Intuitively, to facilitate label pruning during indexing, the hierarchy needs to be built in a top-down manner, i.e., first performing Algorithm 1 without pruning (line 11) for global hubs to generate H-2; then performing the actual Algorithm 1 to build H-1; lastly, remaining miss cases are stored to H-2.

Hierarchy-2. In our implementation, the H-2 construction is performed prior to H-1, offering an additional condition for BFS pruning and cache saving. As demonstrated in [35], the bit-parallel 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.

Hierarchy-1. In Algorithm 1, 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.

Hierarchy-0. Examining Property 2, 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 in Algorithm 2, 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.

A.3 Discussion on Querying

Conversely, to ensure fast query speed, Algorithm 3 is performed in a bottom-up sequence with early termination: for each node pair, if the distance is cached in H-0, then it is the exact SPD and can be immediately returned; otherwise, distances from H-1 and H-2 are calculated, and the smallest one is chosen as the SPD answer. Performing the pair-wise query for all node pairs in $\mathcal{S}(r)$ naturally leads to the sampled subgraph as well as the SPD matrix.

B Detailed Comparison with Existing Models

B.1 Kernel-based GTs

- **GraphGPS** [7] combines rich positional and structural encodings with interleaved local message-passing and linear global attention to achieve both expressivity and linear complexity in node and edge counts. However, its preprocessing stage (e.g., computing structural encodings) incurs an $O(N^3)$ time complexity.
- **Exphormer** [19] leverages virtual nodes and expander graphs to augment the input graph for GT training. The expander graph adds random edges and preserves certain properties of

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 $\mathcal{Q} \leftarrow \{(r, 0)\}$,  $(q(r), \mathcal{M}_r^1(r), \mathcal{M}_r^0(r)) \leftarrow (0, \varnothing, \varnothing)$
7      while $\mathcal{Q} \neq \varnothing$ do
8          $\mathcal{Q}^1 \leftarrow \varnothing$,  $\mathcal{Q}^0 \leftarrow \varnothing$
9          Pop the first element $(v, b(v, r))$ from $\mathcal{Q}$
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 $\mathcal{Q}$
17                     $\mathcal{Q}^1 \leftarrow \mathcal{Q}^1 \cup \{(u, v)\}$, $q(u) \leftarrow q(v) + 1$
18                 else if $q(u) = q(v)$ then
19                     $\mathcal{Q}^0 \leftarrow \mathcal{Q}^0 \cup \{(u, v)\}$
20         for all $(u, v) \in \mathcal{Q}^0$ do
21             $\mathcal{M}_r^0(u) \leftarrow \mathcal{M}_r^0(u) \cup \mathcal{M}_r^1(v)$
22         for all $(u, v) \in \mathcal{Q}^1$ 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 $v \in \mathcal{V}$
```

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 $v \in \mathcal{V}$
```

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 $\boldsymbol{B}_r$
```

the graph structure. Its GT component is mainly built upon GraphGPS, sharing the similar computational complexity.

- **NodeFormer** [6] introduces a kernelized Gumbel-Softmax operator to enable linear-time all-pair message passing for scalable graph structure learning.
- **DIFFormer** [8] proposes an energy-constrained diffusion framework that derives closed-form optimal diffusivity to build a diffusion-based Transformer encoder.
- **PolyNormer** [12] presents a polynomial-expressive graph Transformer that learns high-degree equivariant polynomials via a linear local-to-global attention scheme.

Overall, all kernel-based models incur per-layer training and inference costs of $O(LnF^2 + LmF)$ and memory $O(nF + m)$, scaling linearly with both nodes and edges. Due to the lack of all-pair attention, they also exhibit stronger inductive bias of graph information and weaker expressive power compared to our method and hierarchical GTs, particularly on heterogeneous graphs. In contrast, our method's computation and memory during training and inference depend only on node features, independent of edge count, which provides markedly better scalability on large, sparse graphs.

B.2 Hierarchical GTs

- **NAGphormer** [9] aggregates multi-hop neighborhood features into a fixed-length token sequence per node (via Hop2Token), enabling true mini-batch Transformer training on large graphs.
- **PolyFormer** [11] builds a hierarchical Transformer by progressively coarsening the graph and exchanging information between levels to capture both local and global structure.
- **ANS-GT** [13] uses an adversarial bandit to adaptively sample informative nodes and a two-stage local/global attention scheme to capture long-range dependencies. its adaptive sampling requires $O(ns^2 + Lm)$ preprocessing, making it costly on large graphs.
- **GOAT** [10] implements approximate global self-attention via low-dimensional projection (K-Means) plus a local sampling module to support both homophilous and heterophilous graphs. however, the projection and codebook updates add extra overhead and approximation error.
- **HSGT** [14] leverages multi-level graph coarsening and dedicated horizontal/vertical Transformer blocks to fuse information across scales. yet it demands hierarchical sampling and historical embedding maintenance, increasing implementation complexity.

Overall, hierarchical GTs typically achieve stronger expressive capabilities than kernel-based models, especially for capturing complex, heterogeneous relationships across multiple scales. However, except for NAGphormer and PolyFormer, whose training and inference costs are independent of edge count, other hierarchical GTs still incur per-layer training and inference costs of $O(LnF^2 + LmF)$ and memory $O(nF + m)$ or even higher overhead, greatly undermines model efficiency when the graph is large. In contrast, our method's computation and memory during training and inference depend only on node features, independent of edge count, offering superior scalability on large, sparse graphs.

B.3 Non-GT GNNs

- **GCNJK** [42] records individual layer representations as parallel channels for the last-layer concatenation and prediction, adding up hierarchical graph topological representations in the spatial domain.
- **MixHop** [43] concatenates multi-hop spatial propagations in each of its message-passing layer. Such aggregation results in expanding width of representations over multiple layers.
- **TFGNN** [46] employs approximation of the graph spectrum by slicing, preserving topological properties and achieving practical streamlined processing.
- **S$_2$GNN** [47] integrates the spatial operator for local information and the spectral operator for global knowledge. The utilization of iterative eigen-decomposition also entails positional encodings to further augment the learning.

- **SGFormer** [44] utilizes a single-layer global attention coupled with a simple GNN to capture all-pair interactions in one propagation step, achieving $O(N + E)$ complexity without any positional encodings, pre-processing, or edge embeddings, but with limited expressive power for complex graph structures. SGFormer offers $O(N + E)$ efficiency but lack the rich positional and structural encodings that give graph Transformers their superior expressivity.

Generally, while these message-passing GNNs offer linear time complexity, different issues may arise. For instance, GNNs using full graph topology for propagation usually entail $O(mF)$ memory overhead, which is less scalable. Similar to GT, this can be addressed by mini-batching with RS, but risks hindering efficacy. Empirically, these GNNs also exhibit more training epochs than GTs before convergence. Consequently, while the epoch time seems shorter, the total training time is on par with GTs.

We highlight that these works are largely orthogonal to HubGT, as our focus is on advancing the capabilities of graph Transformer models, rather than benchmarking against message-passing GNN architectures.

B.4 Scalable and Heterophilous Convolutional GNNs

The scalability issue has been extensively examined for convolutional Graph Neural Networks (GNNs) [48, 49], featuring the decoupled processing on some of the largest graph datasets with linear or even sub-linear complexity [50, 51, 52, 53]. Graph simplification techniques including sampling [54, 55, 56, 57] and coarsening [58, 59, 60] are also explored for reducing the graph scale at different hierarchy levels. 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.

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 [26, 27, 61] or augmenting the graph topology [28, 25]. A recent work [62] also employs the idea of hub labeling to refine graph convolution.

B.5 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 [35, 34] 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.

C Detailed Experiment Settings

Dataset Details. Table 4 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 [41], 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 Appendix D.3. For subgraph sampling, we perform parameter search for relative ratio of in/out neighbors represented by s_{out} in rage $[0, 48]$.

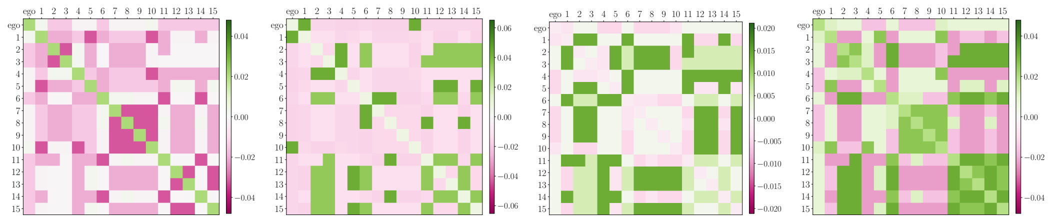
For network architectural hyperparameters, we use $L = 4$ Transformer layers with $N_H = 8$ heads and $F = 128$ hidden dimension for our HubGT model across all experiments. The dropout rates for inputs (features and bias) and intermediate representation are 0.1 and 0.5, respectively. The AdamW optimizer 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.

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 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.

D Additional Experiments

D.1 Empirical Observations

Visualization of Layer-wise Attention Bias. Corresponding to Figure 2, Figure 6 presents the PE bias matrices $\boldsymbol{P}[u,v] = f_B^{(L)}(\boldsymbol{B}_r[u,v])$ of the same SPD matrix $\boldsymbol{B}$ learned by all layers L in the GT model. The encodings are able to exhibit distinct values throughout GT layers, signifying the shift of receptive fields for the in-batch attention to nodes in different distances: in initial layers, node identities are highlighted; in higher layers, clusters of similar neighbors are captured.



(a) $L=0$ (b) $L=1$ (c) $L=2$ (d) $L=3$

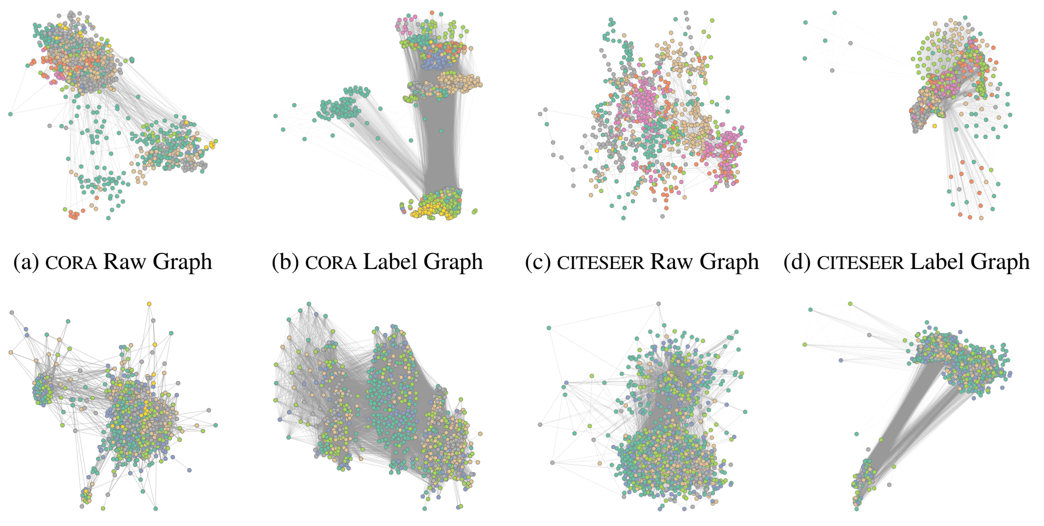
Figure 6: SPD PE as attention bias in different GT layers on CITESEER.

Visualization of Graph Structures. An exemplary illustration of 4 small graphs is displayed in Figure 7. The original CHAMELEON graph is heterophilous, i.e., connected nodes frequently belong to distinct classes. In Figure 7(e), different classes are mixed in graph clusters, which pose a challenge for GTs to perform classification based on edge connections. In contrast, nodes in the graph marked

Table 4: 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.

Heterophily	**Dataset**	**Nodes** n	**Edges** m	F	N_c	**Train**
Homophily	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%
Heterophily	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%

by graph labels in Figure 7(f) clearly form multiple densely connected clusters, exhibiting a distinct hierarchy. Certain classes can be intuitively identified from the hierarchy, which empirically demonstrates the effectiveness of our utilization of graph labeling.


(a) CORA Raw Graph (b) CORA Label Graph (c) CITESEER Raw Graph (d) CITESEER Label Graph

(e) CHAM. Raw Graph (f) CHAM. Label Graph (g) SQUIRREL Raw Graph (h) SQUIRREL Label Graph

Figure 7: Visualization of the hierarchy of original and label graphs on realistic datasets. Color of each node denotes its class.

D.2 Main Experiments on Small Graphs

Table 5 presents the main effectiveness and efficiency evaluations on 6 relatively small datasets. Discussion on the results are in Section 6.2.

Table 5: Effectiveness and efficiency results on small-scale 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.

Visualization in Figure 4. The relative score for each metric is computed as the percentage with respective to the best record in each dataset, i.e., the fastest efficiency and highest accuracy. Unavailable entries are regarded as 0% scores. Efficiency metrics contain the following perspectives and are computed in logarithm: "**Epoch**" and "**Infer**" are average training epoch time and inference time, respectively. Overall "**Learn**" time is computed by the sum of precomputation time and total

Table 6: Effect of different search orders for hub labeling and HubGT learning outcomes on the homophilous CORA and the heterophilous CHAMELEON. Presented results include average index size (i.e., the number of labels) for each node, precomputation time (in seconds), and accuracy (in %). The best result for each column is marked in **bold**.

Centrality	CORA				CHAMELEON			
	H-1	H-0	Pre.	Acc	H-1	H-0	Pre.	Acc
Closeness	9.9	12.9	0.03	**86.0**	4.7	6.5	0.02	**39.9**
PageRank	**2.3**	**2.4**	0.04	85.8	**1.1**	**1.1**	0.02	36.0
Degree	2.4	2.5	**0.02**	**86.0**	1.3	1.3	**0.02**	38.9

Table 7: Effect of the maximum subgraph sample size s on the label graph based on CORA and PHYSICS. Presented results include the average degree after labeling (d_{after}), clustering coefficients before and after hub labeling (C_{before}, C_{after}), index sizes (average number of labels), and total index file size (in MB).

Sample Size	CORA						PHYSICS					
	d_{after}	C_{before}	C_{after}	H-1	H-0	Size	d_{after}	C_{before}	C_{after}	H-1	H-0	Size
16	3.90	0.241	0.648	2.4	2.5	6.2	14.38	0.377	0.302	22.6	135.0	118
32	3.90	0.241	0.621	2.4	2.5	6.2	14.38	0.377	0.306	40.6	230.9	147
48	3.90	0.241	0.584	2.4	2.5	6.2	14.38	0.377	0.300	55.9	294.6	170

training time. The overall efficiency in Figure 4(a) represents overall learning time, inference speed, and memory utilization.

D.3 Effect of Hyperparameters

Search Order. In Table 6, we present experimental results of Algorithm 1 BFS based on different metrics such as closeness centrality and PageRank. It can be observed that, closeness centrality results in larger average label sizes for both H-1 and H-0 indices, which is in line with the observation in [35]. Degree and closeness centrality lead to similar GT accuracy, while PageRank is less representative for selecting useful hubs.

As noted in Appendix A.1, the main consideration in hub selection is indexing efficiency. While other measures may be useful, there is additional overhead for computing these scores, which makes them less applicable to large graphs. Hence, we still recommend degree centrality, which can be efficiently acquired.

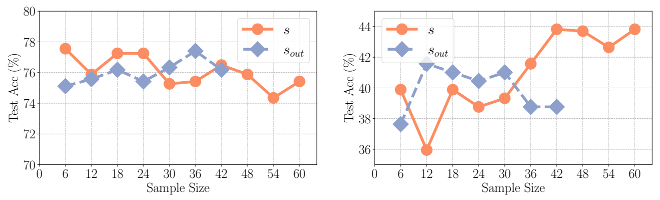
Subgraph Size in Indexing. Table 7 presents more details with varying s when constructing labels. We compute the average clustering coefficient C before and after hub labeling, showing that hub labeling can flatten the node distribution by increasing the clustering property on less clustered graphs such as CORA. Conversely, for highly clustered graphs and larger label sizes, C is slightly reduced, as additional label connections make the graph denser with a more dispersed edge distribution.

For small graphs such as CORA, the average H-1 label sizes $|\mathcal{L}| + |\mathcal{L}'|$ do not vary with different upper bounds of s, which implies that the hubs are sufficiently connected to graph nodes. On the larger PHYSICS, increasing the small $s = 24$ can increase the actual label size, effectively incorporating more remote nodes to form the subgraph. However, the increase is less significant for $s > 48$, which indicates that the hubs are already sufficiently connected. As the H-0 label size $|\mathcal{I}|$ is based on H-1 construction, it also exhibits less increase with larger s, as the larger H-1 is able to cover more SPD pairs and suppress the need for H-0. For most configurations, the H-0 size is significantly smaller than the theoretical bound $O(s^2)$, ensuring empirical space efficiency.

Subgraph Size in Querying. We then study the effectiveness of the label graph hierarchy in HubGT featuring the subgraph generation process in Figure 8, 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 8(a) and 8(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}.



(a) CITESEER (b) CHAMELEON

Figure 8: Effect of sample sizes on different datasets.

Alternative Transformer Architecture. Thanks to the decoupled design, HubGT is able to seamlessly integrate with alternative efficient Transformer architectures. To demonstrate, we implement FlexAttention [63] to replace the standard Transformer (MHA). The results are presented in Table 8. Overall, FlexAttention achieves a 10–20× speedup and significant reductions in GPU memory footprint, verifying the benefits of improving the Transformer architecture.

Table 8: Effectiveness and efficiency results for alternative architectures.

Model	CORA				GENIUS			
	Epoch	Infer	Mem.	Acc	Epoch	Infer	Mem.	Acc
HubGT+MHA	0.20	0.018	2.5	86.0	23.7	2.58	14.3	89.8
HubGT+FlexAttn	0.14	0.013	1.8	86.9	21.5	2.15	1.5	89.1

D.4 Evaluation on Random Graphs

Conventionally, hub labeling is developed for realistic graphs such as road networks and social networks, which assume properties such as sparsity and certain locality/hierarchy. The application to random graphs is usually different and signified by considerable dispersed connections across different local structures. In Table 9, we present a preliminary study to evaluate the effectiveness of our HubGT labeling on different synthetic graphs with $n = 1000$ nodes, including BARABASI-ALBERT and ERDOS-REYNI implemented by PyG [64], CSBM [65], and GENCAT [66].

For BA, ER, and CSBM graphs, both H-1 and H-0 label sizes are relatively small, and there are extreme cases where no H-1 and H-0 labels are required when the edge probability p is high and the graph is dense. These cases imply that the H-2 computation is especially effective, and a fixed number of global hubs are sufficient for SPD computation. In fact, for most node pairs, there is $SPD(u, v) = 1$ or 2 when the raw graph is sufficiently connected, implying that it is already a 2-hop cover. GENCAT is a graph generator closer to realistic graphs with power-law degree distribution, where larger α indicates a "steeper" distribution with more hubs and less long-tailed nodes. In this case, the label sizes are similar to realistic graphs, where more hubs result in smaller H-1 and larger H-0.

Table 9: Effectiveness and efficiency results on random graphs with varying synthesis parameters. Presented results include average index size (i.e., the number of labels) for each node and precomputation time (in seconds).

Dataset	Param.	H-1	H-0	Pre.	Param.	H-1	H-0	Pre.
CSBM	$p = 0.01, q = 0.0$	30.1	62.3	0.02	$p = 0.1, q = 0.1$	0.0	0.0	0.02
ERDOS-RENYI	$p = 0.01$	1.6	2.1	0.02	$p = 0.1$	0.0	0.0	0.02
BARABASI-ALBERT	$d = 10$	1.7	1.7	0.02	$d = 100$	0.5	0.6	0.06
GENCAT	$\alpha = 3.0$	30.6	72.7	0.04	$\alpha = 2.0$	32.8	67.3	0.03

In summary, as noted in Properties 2 and 3, the hierarchy can naturally emerge during the hub labeling process. For random graphs with dispersed/non-local connections, the pattern can be effectively utilized by H-2 to compute SPD.

E Discussions

E.1 Algorithmic Generality, Limitation, and Future Direction

Dynamic Graph and Update. Hub labeling for dynamic graphs has been studied in previous works such as [67]. In brief, it prevents full index updates and only updates the affected labels on candidate paths. Nonetheless, we consider it a promising direction to integrate dynamic hub labeling with GT learning to better handle graph changes.

Multi-device Computation. As HubGT employs the fully decoupled architecture, it renders the future possibility to deploy multi-device *training*: once the precomputation is completed, data parallelism can be employed with multiple Transformers, each handling a batch of tokens, since node tokens are mutually independent and SPD querying is highly local in storage. One potential limitation may be related to the large H-0 index, rendering index duplication less efficient. In this case, HubGT is superior to previous models, especially kernel-based ones, since only node-wise inputs are required and can be easily batched. With respect to multi-device settings for *indexing* computation across distributed graph partitions, the index $\mathcal{L}$ can be divided among devices in a node-wise manner, while the index $\mathcal{I}$ and attributes X need to be mirrored or synchronized throughout the computation.

E.2 Broader Impact

As our work primarily focuses on theoretical contributions to improve the scalability of graph neural networks, we do not foresee it having a direct negative social impact.