

GRAPH TRANSFORMERS FOR LARGE GRAPHS

Anonymous authors

Paper under double-blind review

ABSTRACT

Transformers have recently emerged as powerful neural networks for graph learning, showcasing state-of-the-art performance on several graph property prediction tasks. However, these results have been limited to small-scale graphs, such as ligand molecules with fewer than a hundred atoms, where the computational feasibility of the global attention mechanism is possible. The next goal is to scale up these architectures to handle very large graphs on the scale of millions or even billions of nodes. With large-scale graphs, global attention learning is proven impractical due to its quadratic complexity w.r.t. the number of nodes. On the other hand, neighborhood sampling techniques become essential to manage large graph sizes, yet finding the optimal trade-off between speed and accuracy with sampling techniques remains challenging. This work advances representation learning on single large-scale graphs with a focus on identifying model characteristics and critical design constraints for developing scalable graph transformer (GT) architectures. We argue such GT requires layers that can adeptly learn both local and global graph representations while swiftly sampling the graph topology. As such, a key innovation of this work lies in the creation of a fast neighborhood sampling technique coupled with a local attention mechanism that encompasses a 4-hop reception field, but achieved through just 2-hop operations. This local node embedding is then integrated with a global node embedding, acquired via another self-attention layer with an approximate global codebook, before finally sent through a downstream layer for node predictions. The proposed GT framework, named **LargeGT**, overcomes previous computational bottlenecks and is validated on three large-scale node classification benchmarks. We report a $3\times$ speedup and 16.8% performance gain on `ogbn-products` and `snap-patents` compared to their nearest baselines respectively, while we also scale **LargeGT** on `ogbn-papers100M` with a 5.9% improvement in performance.

1 INTRODUCTION

Transformer networks (Vaswani et al., 2017) have revolutionized representation learning in various domains, particularly in the field of natural language processing (Devlin et al., 2018; Liu et al., 2019; Yang et al., 2019; Raffel et al., 2020; Brown et al., 2020; Dosovitskiy et al., 2020; Touvron et al., 2023; OpenAI, 2023). Their unique ability to model intricate all-pair dependencies in sequential data (or sets of data tokens) has sparked interest in extending Transformer architectures beyond just sequential data, leading to promising research in the area of graph representation learning (Zhang et al., 2020; Dwivedi & Bresson, 2021; Kreuzer et al., 2021; Mialon et al., 2021; Ying et al., 2021; Rampásek et al., 2022; Shirzad et al., 2023). However, these advances are not without their challenges. As graph-based learning tasks grow more complex and the scales of the graph data increase, the limitations of current Graph Transformer (GT) architectures become increasingly evident (Zhao et al., 2021; Chen et al., 2022; Kong et al., 2023).

On the other hand, traditional message-passing neural networks (MPNNs), including variants like GCN (Kipf & Welling, 2016), GAT (Veličković et al., 2018), and GatedGCN (Bresson & Laurent, 2017), function effectively on small-scale graphs such as molecular structures, operating in the order of $O(E)$, or $O(N)$ for sparse graphs (Gilmer et al., 2017). However, their efficiency rapidly decreases when applied to larger graphs with even less than a million nodes. This is primarily because MPNNs consider all neighbors during their aggregation and update steps. To mitigate this, some approaches use neighborhood sampling (NS) to limit the number of sampled neighbors

for each node up to a certain number of hops (Hamilton et al., 2017). While this method keeps computational costs in check, it encounters intractability issues when the graph scales to hundreds of millions of nodes or more. Moreover, even if NS is computationally feasible for two or three hops, the MPNNs become confined to capturing only highly localized information, which might be insufficient for tasks on large graphs where more global context is crucial (Lim et al., 2021; Dwivedi et al., 2022; Kong et al., 2023).

Graph Transformers (GTs) could provide a potential solution, given their ability to model long-range dependencies and attend to global neighborhoods (Rampásek et al., 2022). However, the intractability remains, given that GTs with all-pair attention, *i.e.*, each node attending to every node, would be quadratic ($O(N^2)$) computationally. When applied with NS, GTs inherit the same limitations, confining their effectiveness to localized regions. Without NS, the task of attending to global neighborhoods necessitates approximation (Wu et al., 2022; 2023; Shirzad et al., 2023) or sampling (Zhao et al., 2021; Zhang et al., 2022; Zhu et al., 2023) for computational feasibility, thus bringing us back to the original challenges of sampling and (in)efficient access to global information.

Present Work. In this paper, we introduce a comprehensive approach to overcoming the aforementioned critical challenges in learning on large-scale graphs by focusing on two essential design principles: model capacity and scalability. We enhance model capacity by integrating both local and global graph information for building node representations, while ensuring scalability with an efficient sampling approach. The proposed framework is summarized as follows:

1. **Framework Design:** We present LargeGT, a new framework that integrates both local and global graph representations while minimizing the computational cost incurred at both the stages. In consistency with recent working recipes in graph learning (Rampásek et al., 2022; Kong et al., 2023) LargeGT utilizes two distinct modules — LOCALMODULE and GLOBALMODULE — to handle local and global information exchange efficiently.
2. **Localized Representations:** Within the LOCALMODULE, we present a novel tokenization strategy that prepares a fixed set of tokens for each graph node to be processed by a Transformer encoder, resulting in rich local feature representations. Importantly, this mechanism leverages a neighborhood sampling approach that consists of an offline sampling stage and incorporates local context features, enabling a broad 4-hop receptive field through just 2-hop operations.
3. **Global Representations:** For the GLOBALMODULE, we implement an approximate codebook-based approach, adapted from Kong et al. (2023), to enable global graph attention with computational complexity linear to the codebook size. This design choice ensures that both modules can operate independently of the graph size, thereby ensuring scalability.
4. **Computational Efficiency:** Our approach effectively mitigates computational bottlenecks traditionally associated with sampling techniques and global information flow. As a result, we enable incorporation of both local and global graph representations without compromising performance.
5. **Empirical Validation:** We validate the competitiveness and scalability of LargeGT with baselines in a scalable setting using ogbn-products, snap-patents and ogbn-papers100M datasets which are among the largest benchmarks with node in ranges 2.5M, 2.9M and 111.1M, respectively. Notably, we obtain a $3\times$ speedup and 16.8% performance gain on ogbn-products and snap-patents compared to their best baselines respectively, while on ogbn-papers100M we scale LargeGT with a 5.9% improvement in performance.

Overall, our work not only investigates on the key challenges and presents design elements for GTs at scale but also provides a robust framework for future research in this area.

2 RELATED WORK

Challenges in MPNN Scaling. When learning on large graphs, the principal issue faced by message-passing based graph neural networks (MPNNs) is the neighbor explosion phenomenon (Hamilton et al., 2017; Zhao et al., 2021), since the neighborhood sets of nodes at successive hops expand exponentially (Alon & Yahav, 2021). Early efforts in scaling MPNNs employed the use of neighbor sampling (NS) to sample neighbors of nodes in a graph recursively that reduces the overall neighborhood sets sending messages for nodes’ feature updates (Hamilton et al., 2017). While this brings a reduction in memory and compute footprint, the MPNN is still intractable to (i) aggregate

information at hops greater than 2 or 3 in very large graphs due to the fact that the the size of the successive neighborhood grows exponentially, and (ii) access global information in the graph, which is also a well-acknowledged limitation for several works along this line (Chen et al., 2017; 2018; Huang et al., 2018; Zeng et al., 2019). Information propagation *prior to* or *after* the training stage (Gasteiger et al., 2018; Wu et al., 2019; Frasca et al., 2020) are also adopted as ways to address the intractability brought by neighborhood explosion, which follow the aforementioned limitations. Several works also propose pre-training (Han et al., 2022) or distillation strategies (Zhang et al., 2021; Guo et al., 2023) to mitigate these impacts.

Graph Transformers and Scalability. The apparent access to global information is a driving factor behind the recent plethora of works on Graph Transformers (GTs) with all pair attention (Ying et al., 2021). We refer to Müller et al. (2023) for a detailed taxonomy and component-wise study of GTs. However, an obvious barrier for GTs to scale to large graphs is the quadratic complexity brought by full-graph attention, *i.e.*, $O(N^2)$, with N being the number of nodes in a graph. The use of sparse Transformers (Rampásek et al., 2022; Shirzad et al., 2023) or approximated global attention Wu et al. (2022; 2023) alleviate this issue to some extent to bring down the complexity to sub-quadratic. Yet, these methods remain unscalable on single large graphs due to the entire graph structure being operated upon. In order to address this limitation, either NS-like computational boundary is enforced for each node (Shi et al., 2020; Zhao et al., 2021) or a fixed length sequence or set is prepared for each node prior to training (Zhang et al., 2020; Chen et al., 2022). Such solutions either inherit the limitations of NS as discussed above, or face infeasibility due to adjacency matrix multiplications as the graph size grows larger.

Clustering based GTs. Orthogonally, several recent works use hierarchical clustering or partitioning to perform global attention on the coarsened or super nodes (Zhang et al., 2022; Zhu et al., 2023). However, the coarsening step remains intractable for very large graphs with sizes in hundreds of millions or more. Finally, Kong et al. (2023) use a combination of NS based local module and a global module consisting of a trainable fixed-sized codebook that represents global centroids. While the sampling limitations remain, the global module based on the centroids is efficient and something that we consider in our proposed approach to compute global representations.

3 RECIPE FOR BUILDING TRANSFORMERS FOR LARGE GRAPHS

In the previous sections, we identified critical limitations in existing graph learning models, specifically their inability to effectively merge local and global graph features when working with very large graphs. While MPNNs struggle with the ‘neighbor explosion’ problem, making it difficult to aggregate information beyond 2-3 hops GTs *can* capture global context, but are hampered by quadratic complexity in full graph attention. These challenges, although formidable, outline the essential criteria for a successful GT model tailored for large graphs. This leads us to present a recipe focusing on model capacity and scalability for building GTs for large graphs. In this section, we first discuss the design principles of model capacity¹ and scalability. These factors play a crucial role in determining the design and feasibility of a Graph Transformer, particularly when it comes to managing extremely large graphs that have node counts in the millions or higher. Finally, following the design characteristics, we introduce our proposed framework — LargeGT.

3.1 DESIGN PRINCIPLES

D1- Model Capacity. A graph learning model should possess the ability to incorporate both local and global information from the original large graph.

Local Inductive Bias: A node’s local connectivity presents a rich source of information that is essential to utilize even in a large graph setting. A straightforward local aggregation of all neighboring nodes followed by update equation in an MPNN is impractical due to which several works utilize sampling techniques (as discussed in Section 2). The incorporation of such local information, while

¹Note that in this work, we do not contextualize a model’s capacity in terms of Weisfeiler-Leman (WL)’s expressiveness (Weisfeiler & Leman, 1968; Morris et al., 2019; Xu et al., 2019) as it is known that almost all of non-isomorphic graphs (or subgraphs) are distinguishable by a 1-WL equivalent model (MPNN) in the presence of node features (Cotta et al., 2021), and we do not consider graphs with anonymous nodes.

remaining in a feasible computational boundary, is a key ingredient when building a graph learning model for larger graphs and helps immensely in homophilic tasks (Ma et al., 2021; Mao et al., 2023).

Access to Non-Local Information: A node’s capability to incorporate features beyond near-local neighbors may be vital for long-range or non-homophilic tasks (Lim et al., 2021; Dwivedi et al., 2022). The sampling strategies discussed in Section 2 fail to allow a model to access nodes’ distant hops or global information, motivating the use of GTs (Rampásek et al., 2022; Shirzad et al., 2023). GTs with all-pair attention alleviate this concern, however not all such mechanisms reviewed in Section 2 are scalable, which is a key concern. The recent global attention mechanism proposed in GOAT (Kong et al., 2023) uses dimensionality reduction scheme serving as a “conceptual” global context for a node to attend when looking for non-local information. Without exhaustive computation, a large graph learning model should allow access to global graph context (Cai et al., 2023).

D2- Scalability. Learning on large graphs become infeasible for several existing models due to hindrances in sampling mechanisms or global information modules (Ying et al., 2021; Rampásek et al., 2022; Zhao et al., 2021). As such, we consider the following factors to be vital for computational feasibility and ensuring scalability of the GTs when graphs grow larger.

Efficient Neighbor Node Set Retrieval: As reviewed in Section 2, despite the use of sampling techniques, it becomes intractable to retrieve neighbor node sets from hops greater than two or three if the graph in consideration is very large (Zhang et al., 2021; Guo et al., 2023). In fact, even a third hop neighbor set retrieval takes a significant amount of time for graphs with hundreds of millions of nodes, as the computational complexity of retrieving a node’s l hop neighborhood is $O(d^l)$, where d is the average node degree. Therefore, for efficient retrieval for large graphs, we establish *only* two hops retrieval as a key constraint; this limit is commonly adopted in large-scale graph learning applications (Sankar et al., 2021; Ying et al., 2018; Tang et al., 2022).

Efficient Global Information Access. As much as the significance of global information is discussed as a key recipe in the Design **D1**, we reiterate that the global information flow should come at an inexpensive cost. There are multiple candidates for an efficient access to global information, such as sparse global attention (Shirzad et al., 2023), use of virtual nodes (Cai et al., 2023) and use of an approximate centroid-based codebook (Kong et al., 2023). Since we focus on large graphs where the former two candidates can be infeasible due to their dependency on the number of nodes in the graph, we aim for incorporating global information without bottlenecks due to a graph’s large size, such as the method implemented in Kong et al. (2023) which depends on a dynamic codebook comprising of global centroid tokens.

Distributed Training. Handling very large graphs will be impractical without taking advantage of the distributed computing infrastructure in which several of the real world graphs occur, *eg.* social networks, which are massive and are generally distributed over different machines. Ensuring efficient run times during both training and inference is vital to allow a GT to handle large graphs within acceptable time frames, keeping the cost feasible and lower. For a model to scale on large graphs, the training steps should do away with the bottlenecks brought by traditional MPNNs and sparse GTs. For instance, MPNNs usually sample nodes and their neighbors during the mini-batching step of the training process, which also applies in recent GTs (Zhao et al., 2021; Kong et al., 2023). In principle, this translates to maintaining the adjacency matrix of a graph in one machine. This can be challenging given an example that a simple 3-layer GraphSAGE with NS requires around 350GB+ RAM on the 111.1M sized ogbn-papers100M following standard configurations of OGB Hu et al. (2020). The single-machine memory requirement can be much larger than the availability on standard machines with on sophisticated models and on industry-scale graphs with billions of nodes and edges (Shi et al., 2023; Ying et al., 2018). Thus, avoiding the need to maintain the entire graph data on a single machine when learning is desirable.

3.2 OUR PROPOSED FRAMEWORK: LARGE GT

Incorporating these desiderata jointly, we next introduce our proposed framework, LargeGT which is designed from first-principles to enable the application of GTs to massive large-scale graphs. We refer to Figure 1 for a sketch of the proposed architecture.

Notations. We denote a given graph with $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with \mathcal{V} being the set of nodes and \mathcal{E} the set of edges, and $N = |\mathcal{V}|$ and $E = |\mathcal{E}|$ being their cardinalities. The graph structure is represented

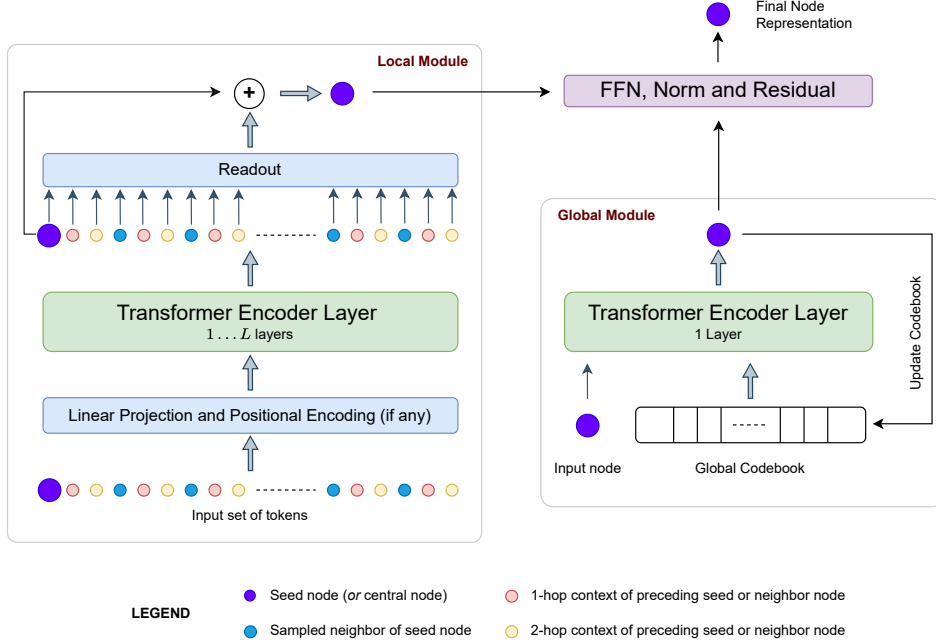


Figure 1: Architectural diagram of LargeGT illustrating the process of updating a node’s representation denoted by the seed node. FFN and Norm denotes Feed Forward Network and Normalization layer respectively. Neighbors are sampled for a central or seed node offline, prior to the training stage using Algorithm LOCALNODES. The feature vectors coming from both local module and global module are concatenated before passing to the ‘FFN, Norm and Residual’ module.

by the adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ where $\mathbf{A}_{ij} = 1$ if there exists an edge between nodes i and j , otherwise $\mathbf{A}_{ij} = 0$. The features for a node i is denoted by $\mathbf{H}_i \in \mathbb{R}^{1 \times D}$ and for all nodes in the graph \mathcal{G} is denoted by $\mathbf{H} \in \mathbb{R}^{N \times D}$. While we assume, for simplicity, that the graph \mathcal{G} has no edge features, these can be incorporated using standard methods as done in the literature, such as using edge features during message passing to update node representations.

Update Algorithm. We now define the equations which update the feature representations of a node using the LargeGT framework. Given a node i with its input feature $\mathbf{H}_i^{\text{in}} \in \mathbb{R}^{1 \times D_{\text{in}}}$, the aim is to obtain its output features $\mathbf{H}_i^{\text{out}} \in \mathbb{R}^{1 \times D_{\text{out}}}$ which can be passed to appropriate prediction heads and/or loss functions, subsequently, depending on the learning task. For simplicity, we will represent $\mathbf{H}_i \in \mathbb{R}^{1 \times D}$ in the following equations.

$$\mathbf{S}_i = \text{LOCALNODES}_i(\mathbf{A}, K) \in \mathbb{R}^{1 \times K} \quad (1)$$

$$\mathbf{X}_i = \text{INPUTTOKENS}_i(\mathbf{S}_i, \mathbf{H}_i^{\text{in}}, \mathbf{C}_i) \in \mathbb{R}^{1 \times 3K \times D} \quad (2)$$

$$\mathbf{H}_i^{\text{local}} = \text{LOCALMODULE}(\mathbf{X}_i) \in \mathbb{R}^{1 \times D} \quad (3)$$

$$\mathbf{H}_i^{\text{global}} = \text{GLOBALMODULE}(\mathbf{H}_i^{\text{in}}) \in \mathbb{R}^{1 \times D} \quad (4)$$

$$\hat{\mathbf{H}}_i = \text{FFN}(\mathbf{H}_i^{\text{local}} \parallel \mathbf{H}_i^{\text{global}}) \in \mathbb{R}^{1 \times D} \quad (5)$$

$$\mathbf{H}_i^{\text{out}} = \mathbf{H}_i^{\text{in}} + \text{NORM}(\hat{\mathbf{H}}_i) \in \mathbb{R}^{1 \times D} \quad (6)$$

where, \mathbf{S}_i reflects the K sampled local nodes from LOCALNODES in Algorithm 1, and \mathbf{X}_i is the set of input tokens of size $3K$ prepared using INPUTTOKENS in Algorithm 2. Additionally, LOCALMODULE consists of a standard Transformer encoder (Vaswani et al., 2017) which could be a stack of multiple layers to produce each token’s representations, with a readout function at the end that converts the set of tokens to one feature vector as sketched in Figure 1, and GLOBALMODULE consists of a single layer Transformer encoder adapted from Kong et al. (2023) that allows a node to attend to an approximate global representation of all nodes in the graph through a projection of all nodes’ features in a codebook of a fixed size, that is updated at each iteration (see Section A.3). \parallel denotes concatenation. $\mathbf{C} \in \mathbb{R}^{N \times 2 \times D}$ is the context feature matrix which provides the 1 and 2 hop neighborhood context for all nodes in the graph and can be precomputed as $\mathbf{C}^0 = \hat{\mathbf{A}}\mathbf{H} \in \mathbb{R}^{N \times 1 \times D}$ and $\mathbf{C}^1 = \hat{\mathbf{A}}^2\mathbf{H} \in \mathbb{R}^{N \times 1 \times D}$ where $\hat{\mathbf{A}}$ is the normalized adjacency matrix (Chen et al., 2022).

Algorithm 1 LOCALNODES: Algorithm to fetch a multiset of local nodes from 1 and 2 hop neighbors for each node.

Require: A graph with adjacency matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, and the size of the multiset K .

Ensure: Return the matrix with K -sized multisets for each node $\mathbf{S} \in \mathbb{R}^{N \times K}$

```

1: Initialize: Multisets for all nodes  $\mathbf{S} \in \mathbb{R}^{N \times K}$ 
2: for  $i = 0$  to  $N - 1$  do
3:    $\hat{\mathbf{T}} \leftarrow$  1 and 2 hop neighbors of node  $i$ 
4:   if  $|\hat{\mathbf{T}}| \geq k - 1$  then
5:      $\mathbf{T} \leftarrow$  Randomly sample  $K - 1$  nodes from  $\hat{\mathbf{T}}$ 
6:   else if  $|\hat{\mathbf{T}}| < K - 1$  and  $|\hat{\mathbf{T}}| > 0$  then
7:      $\mathbf{T} \leftarrow$  Randomly sample from  $\hat{\mathbf{T}}$  with replacement to make  $K - 1$  nodes
8:   else
9:      $\mathbf{T} \leftarrow$  Randomly sample  $K - 1$  nodes from  $\{0, 1, 2, \dots, N - 1\}$ 
10:  end if
11:   $\mathbf{S}_i \leftarrow \{i\} + \{\mathbf{T}\} \in \mathbb{R}^k$ 
12: end for

```

Algorithm 2 INPUTTOKENS: Algorithm for Mini-Batch Preparation for Local Module

Require: Mini-batch of M samples $\mathbf{S} \in \mathbb{R}^{M \times K}$ where K is the total size of the multiset of nodes for each node, Feature matrix $\mathbf{H} \in \mathbb{R}^{N \times D}$, Hop context features $\mathbf{C} \in \mathbb{R}^{N \times 2 \times D}$.

Ensure: Return the input data of all nodes in mini-batch $\mathbf{X} \in \mathbb{R}^{M \times 3K \times D}$.

```

1: Initialize:  $\mathbf{X} \in \mathbb{R}^{M \times 3K \times D}$ 
2: for  $i = 0$  to  $M - 1$  do
3:   for  $j = 0$  to  $3K$  with step 3 do
4:      $\mathbf{X}_{i,j} \leftarrow \mathbf{H}[\mathbf{S}_{i,j}]$  # node feature for node  $i$  from the feature matrix  $\mathbf{H}$ 
5:      $\mathbf{X}_{i,j+1} \leftarrow \mathbf{C}[\mathbf{S}_{i,j}, 0]$  # 1 hop context feature for the node  $i$  from  $\mathbf{C}$ 
6:      $\mathbf{X}_{i,j+2} \leftarrow \mathbf{C}[\mathbf{S}_{i,j}, 1]$  # 2 hop context feature for the node  $i$  from  $\mathbf{C}$ 
7:   end for
8: end for

```

Finally, FFN refers to a feed forward network used in Transformers, and NORM is normalization, which can be either of LayerNorm (Ba et al., 2016) or BatchNorm (Ioffe & Szegedy, 2015). Additionally, we note here that the Algorithms 1 and 2 are defined for either all nodes or a mini-batch of nodes, while their respective outputs can be accessed index-wise in the above Eqns. 1 and 2.

Offline Step Prior to Training. The Algorithm 1 LOCALNODES is run on CPU prior to the training to sample local nodes for each node in the graph, and can be parallelized to multiple cores or machines. The parallelization is possible due to the fact that the fetch of 1-hop and 2-hop neighbors in Step 3 and the subsequent steps of Algorithm 1 is independent for each node and can be executed on different CPU cores in parallel. It’s also worth mentioning that the graph is not required to be stored on a single machine’s memory for this step. In line with our design principles, the graph can be distributed across multiple machines using techniques such as key-value stores or graph databases to circumvent the issue of memory limitations. This ensures that the offline step aligns with the scalability considerations of handling very large graphs.

Mini-Batching for Local Module. The Algorithm 2 abstracts a mini-batching stage during the training, which prepares the input tokens for each node in the mini-batch with M node samples. This process contains a nested loop that can also be heavily parallelized across available cores. The mini-batch algorithm is implemented in a dataloader’s preparation function, such as the PyTorch DataLoader collate (Paszke et al., 2019). Besides, an important feature provided by INPUTTOKENS is that it *increases* the receptive field of a node *up to 4 hops*, based on just 2-hop computations. To illustrate this, consider a node with its K sized sampled set of 1-hop and 2-hop neighbors. Since the steps 5-6 in Algorithm 2 allows for a nodes’ 1-2 hop neighbors to retrieve *their* 1-hop and 2-hop neighborhood context, the node i in consideration will have access to information from up to 4 hops. We have included further elaboration of this feature in Figure A.2 in Section A.2.

Characteristics of the Framework. The LargeGT framework fulfills the desired characteristics (D1-D2, Section 3.1) for a graph learning model to scale on large graphs. **D1-Model Capacity:** By incorporating both a local and global module, LargeGT can build node representations using

information from local neighborhoods as well as global graph context. The local module leverages offline neighbor sampling and local context features to efficiently aggregate local structure up to 4 hops while the global module allows attending to the entire graph through a trainable codebook. **D2-Scalability:** The framework only samples up to 2-hop neighbors for each node, ensuring efficient neighbor set retrieval. The runtime is also efficient as the local module operates on sampled neighbors and the global module utilizes a fixed size codebook, which is already efficient as demonstrated in Kong et al. (2023). Similarly, the offline neighbor sampling step allows the local node sets and neighborhood contexts to be prepared independently of the graph structure, essentially converting the graph learning task into a standard neural network training problem on the sampled nodes and contexts. This allows easy parallelization across machines like for other modalities such as image and text, alleviating bottlenecks caused in traditional GNN training that require adjacency matrix access on each machine, in principle. The input token preparation is also independent of the graph structure. In summary, LargeGT satisfies the key design principles of model capacity and scalability to effectively scale on large graph datasets. The local and global components allow it to learn both from local structure as well as model global dependencies in the graph, without any *explicit* bottleneck caused due to the size of a graph. In the next section, we will demonstrate that this model is not only scalable, but offers strongly competitive performance.

Complexity. The computational complexity of the LOCALMODULE in LargeGT is $O((3K)^2)$, while that of the GLOBALMODULE is $O(B)$ where K is the size of the local nodes, $3K$ is the size of the tokens for the LOCALMODULE and B is the size of the codebook in GLOBALMODULE. As such, the framework is not bottlenecked with the size of nodes in the graph, as in prior literature. Note that the complexity of Algorithm 1, which is a one-time offline step, does not affect the computational complexity of LargeGT.

4 EXPERIMENTS

4.1 DATASETS AND EXPERIMENTAL SETUP

Datasets. We evaluate the proposed LargeGT model architecture on single large graph datasets with node classification tasks. Since we particularly focus on large graphs, we do not conduct evaluations on smaller benchmarks with lesser than million nodes as the limitations of scaling a model are not present. We use `ogbn-products` and `snap-patents` for our model prototyping and experiments, while we also scale our model on `ogbn-papers100M`. The datasets’ summary is presented in Table 1. **ogbn-products** is a dataset containing Amazon co-purchasing network (Bhatia et al., 2016) with nodes representing products and edges representing the products being purchased together. We use the Open Graph Benchmark (Hu et al., 2020) version of the dataset with their standard splits and available features. **snap-patents** is a network of US patents (Leskovec & Krevl, 2014) where each node represents a patent and an edge represents patent nodes which cite each other. We use the `snap-patents` dataset from Lim et al. (2021) with their default splits and features. Finally, we use **ogbn-papers100M** dataset (Hu et al., 2020) which is one of the largest publicly available single large graph benchmarks. Nodes in the graph denote an arXiv paper while directed edges denote papers which cite other papers (Wang et al., 2020). As with the former dataset, we use the default splits and features provided by OGB. Among the three datasets, `snap-patents` is a non-homophilic dataset, while the rest are homophilic.

Scalable Baselines. We design our experiments in a way to ensure scalability with efficient neighbor node set retrieval (D2, Section 3.1) on large graphs, with sizes even beyond the benchmarks we use for the demonstration in this work. As such, we use constrained versions of existing MPNNs or GT baselines, denoted by Model- δ and call them ‘Scalable Baselines’. We select GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), GT-sparse (Shi et al., 2020; Dwivedi & Bresson, 2021), NAGphormer (Chen et al., 2022) and GOAT (Kong et al., 2023) as the baselines which encompass fundamental GNNs as well as recent scalable GTs. Following D2, we constrain the graph propagation related operations in all these baselines to 2 hops only. The goal of our exper-

Table 1: Summary of the datasets used in our experiments.

Dataset Name	Total Nodes	Total Edges	Node Feats	Class Size	Class Label
<code>ogbn-products</code>	2,449,029	61,859,140	100	47	product category
<code>snap-patents</code>	2,923,922	13,975,788	269	5	time granted
<code>ogbn-papers100M</code>	111,059,956	1,615,685,872	128	172	subject area

Table 2: Results for `ogbn-products`, `snap-patents` and `ogbn-papers100M` datasets. All results reported are on 4 runs. `LargeGT` as well as models with `- δ` suffix are the versions of the respective original architecture with `D2`, *i.e.*, only upto 2-hop computations. GraphSAGE, GAT and GT-sparse use NS with sizes [20, 10]. Colors denote **First**, **Second** and **Third**. Higher is better.

(a) <code>ogbn-products</code>		(b) <code>snap-patents</code>		(c) <code>ogbn-papers100M</code>	
Model	Test Acc	Model	Test Acc	Model	Test Acc
GraphSAGE- δ	76.62 \pm 0.93	GraphSAGE- δ	48.43 \pm 0.21	GOAT-full- δ	61.12 \pm 0.10
GAT- δ	77.38 \pm 0.59	GAT- δ	45.92 \pm 0.22	LargeGT-full	64.73\pm0.05
GT-sparse- δ	60.76 \pm 0.00	GT-sparse- δ	47.81 \pm 0.00		
NAGphormer- δ	75.28 \pm 0.04	NAGphormer- δ	60.11\pm0.05		
GOAT-local- δ	81.17\pm0.12	GOAT-local- δ	40.95 \pm 0.16		
GOAT-global- δ	70.28 \pm 1.95	GOAT-global- δ	42.65 \pm 0.07		
GOAT-full- δ	79.88\pm0.20	GOAT-full- δ	50.28 \pm 0.14		
LargeGT-local	78.95\pm0.80	LargeGT-local	68.19\pm3.11		
LargeGT-full	79.81\pm0.25	LargeGT-full	70.21\pm0.12		

imental setup is to show how the scalability constraints affect existing models’ capabilities, which can be addressed by `LargeGT`; for this reason, we do not use enhanced techniques for obtaining top leaderboard results such as auxiliary label propagation (Shi et al., 2020; Huang et al., 2020) or augmentations (Kong et al., 2020). We apply similar restrictions in our proposed `LargeGT` as well. For hyperparameters, architectural and training setup of the baselines, we adopt the hyperparameters available in the original model papers or the OGB examples repository (see Section A.5). We refer to Section A.4 for a comparison of scalable and original baselines.

LargeGT Models. We train and evaluate two versions of `LargeGT` in our experiments, denoted as `LargeGT-local` and `LargeGT-full`. In local version we omit the `GLOBALMODULE` (Eqn 4) and only use local representations in Eqn. 5. In the full version, we use both the local and global modules and follow the entire formulation as in Eqns. 1-6. We implement single layer of Transformer encoder in the local and global modules.² Other hyperparameter details are included in Section A.5.

4.2 NUMERICAL RESULTS AND DISCUSSION

We now present the numerical results in this section. Table 2 presents the main comparison of the baselines `Model- δ` with the `LargeGT` models. For `ogbn-papers100M`, we only report the best performing baseline on the homophilic task of `ogbn-products` due to computational constraints. Figure 2 shows the training time per epoch incurred by the models, while Figure 3 shows the sensitivity analysis of the number of nodes sampled (K) in Algorithm 1 which is used by `LOCALMODULE` in `LargeGT`. We present our observations as follows.

On Performance. On comparison with the scalable baselines in Table 2 consisting of both GNNs and GTs, we observe `LargeGT` to attain competitive performance to the best baseline `GOAT-local- δ` on `ogbn-products`, while on `snap-patents`, `LargeGT` beats all the baselines. The lower scores of `GraphSAGE- δ` and `GAT- δ` models reveal, in part, how the constraint `D2` brings down the model capacity and such a network with a 2-hop graph receptive field may not incorporate necessary information required to build useful node representations. Among all the models on `ogbn-products`, `GOAT-local- δ` is the best performing, which on `snap-patents` ranks the lowest, understandably due to the latter’s non-homophilic characteristic where local-only information aggregation may not be enough. All results considered, `LargeGT` ranks among the best models on both the datasets, suggesting the capacity to work well in both homophilic and non-homophilic settings. Finally, we report the results of `LargeGT-full` on `ogbn-papers100M` together with `GOAT-full- δ` which is the best performing baseline on a similar homophilic task `ogbn-products` on a computational budget of 48h. We observe `LargeGT-full` to be significantly better (5.9%) in performance compared to `GOAT-full- δ` , hence showing the ability to scale to massive graphs.

On Design Characteristics. The scores of `LargeGT` indicate the need to incorporate both local and global neighbor information, as well as the suitability of such architecture to perform both in homophilic and non-homophilic tasks. The absolute gain in performance of `LargeGT-full` against `LargeGT-local` which is around 1% and 2% respectively for `ogbn-products` and `snap-patents` demonstrates how `LargeGT` satisfies `D1` design characteristic. `D2` which ne-

²We observed decreased performance when stacking multiple layers, in consistent with recent works such as Chen et al. (2022); Kong et al. (2023), and leave the exploration of multi-layered version for later.

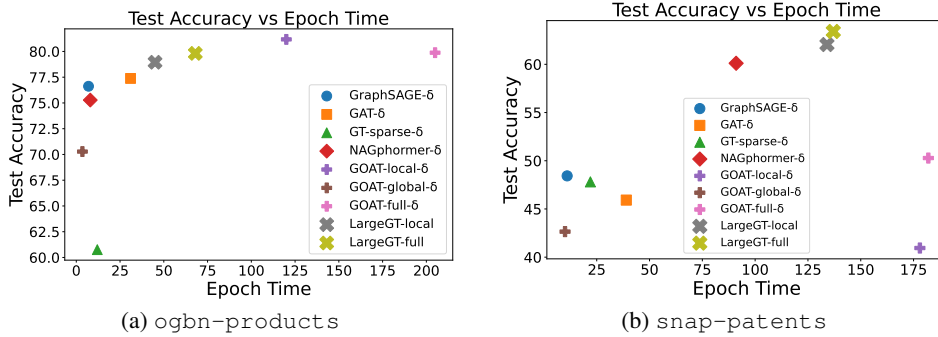


Figure 2: Training time per epoch of various baselines with the proposed architecture. LargeGT reports relatively faster epoch times compared to the best baselines despite performing competitively.

cessitates the computational feasibility of a network helps keep the epoch times of each baseline feasible, along with LargeGT. See the epoch times of each experiments in Figure 2. If we further compare GOAT-full-δ with the original GOAT-full (NS of 3 hops) (Kong et al., 2023), we report per epoch times of 205s and 497s respectively on ogbn-products, which suggests how existing works without D2 can be computationally expensive. Finally, we ensure parallelizability in principle given that the Algorithms 1-2 effectively convert the training problem on a single large graph to a standard neural network training on the sampled tokens, which can leverage parallelization without the need of a single global graph context maintained in the memory of any one machine.

On Runtime. In Figure 2, we present the training time per epoch of all the models considered. We observe the epoch time of LargeGT to be faster up to a maximum of 3× times (GOAT-full-δ vs. LargeGT-full). Further, Fig. A.1 in Sec. A.1 shows better learning and generalization curves of our proposed architecture against the baselines.

On Node set sizes K . In Figure 3, we show results of LargeGT-full experiment on an important hyperparameter in our proposed framework, the number of nodes to be sampled from LOCALNODES (Algorithm 1). The value of K is chosen from a list of [20, 40, 50, 60, 80, 100, 150, 200]. We observe best performance on $K = 100$ and $K = 50$ for ogbn-products and snap-patents respectively, which somewhat aligns with the extent of their tasks being non-homophilic since a lower K would reflect lesser tokens from the local neighborhood.

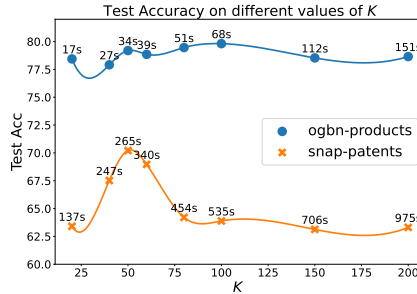


Figure 3: K hyperparameter analysis of Ours-full model on ogbn-products and snap-patents datasets. Labels on the data points denote the training time per epoch.

5 CONCLUSION

In this work, we propose the LargeGT framework for designing Graph Transformers on very large graphs, such as those with at least millions nodes. We studied the characteristics and constraints necessary for a learning model to effectively scale on large graphs, namely model capacity, computational feasibility, and distributed training ability. We design LargeGT to satisfy the intended criteria, and show a proof of concept of how these components can be incorporated into a GT model to obtain competitive results in an improved runtime compared to architectures in prior literature.

Further research. In our LargeGT framework, we focus on key design elements that offer both a robust model and the ability to scale on large graphs. We believe some features of this framework are worth further study. The main goal of this work is to demonstrate the core components like the sampling and tokenization steps (LOCALNODES, INPUTTOKENS), as well as the update modules (LOCALMODULE, GLOBALMODULE). However, these can be further optimized for specific types of graphs. For instance, our current tokenization approach allows a wide 4-hop receptive field through simpler 2-hop operations. Yet, other sampling techniques could be more efficient in capturing complex relationships. Same applies for exploration on GLOBALMODULE’s efficiency.

REPRODUCIBILITY STATEMENT

We build LargeGT primarily on PyTorch (Paszke et al., 2019) leveraging PyTorch Geometric (Fey & Lenssen, 2019), DGL (Wang et al., 2019) and their dependent open-source libraries to perform some of the graph processing steps. We also make use of the codebase generously released by Kong et al. (2023) as we adapt some of their functionalities. While we will release the code repository of our work after this manuscript is peer-reviewed, we provide important dataset information and experimental setup in Section 4 and include necessary hyperparameter information in Section A.5.

REFERENCES

- Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. In *International Conference on Learning Representations*, 2021.
- Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- K Bhatia, K Dahiya, H Jain, A Mittal, Y Prabhu, and M Varma. The extreme classification repository: Multi-label datasets and code. url: <http://manikvarma.org/downloads/xc.XMLRepository.html>, 2016.
- Xavier Bresson and Thomas Laurent. Residual Gated Graph ConvNets. *arXiv:1711.07553*, 2017.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Chen Cai, Truong Son Hy, Rose Yu, and Yusu Wang. On the connection between mpnn and graph transformer. *arXiv preprint arXiv:2301.11956*, 2023.
- Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. *arXiv preprint arXiv:1710.10568*, 2017.
- Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.
- Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*, 2022.
- Leonardo Cotta, Christopher Morris, and Bruno Ribeiro. Reconstruction for powerful graph representations. *Advances in Neural Information Processing Systems*, 34:1713–1726, 2021.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.
- Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.
- Vijay Prakash Dwivedi, Ladislav Rampásek, Mikhail Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Neural Information Processing Systems (NeurIPS 2022), Track on Datasets and Benchmarks*, 2022.
- Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.
- Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

- Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018.
- Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.
- Zhichun Guo, William Shiao, Shichang Zhang, Yozen Liu, Nitesh V Chawla, Neil Shah, and Tong Zhao. Linkless link prediction via relational distillation. In *International Conference on Machine Learning*, pp. 12012–12033. PMLR, 2023.
- Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.
- Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. Mlpinit: Embarrassingly simple gnn training acceleration with mlp initialization. In *The Eleventh International Conference on Learning Representations*, 2022.
- Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *34th Conference on Neural Information Processing Systems*, 2020.
- Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations*, 2020.
- Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems*, 31, 2018.
- Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.
- Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv:1609.02907*, 2016.
- Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. 2020.
- Kezhi Kong, Jiu hai 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*, 2023.
- 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.
- Jure Leskovec and Andrej Krevl. Snap datasets: Stanford large network dataset collection, 2014.
- Derek Lim, Xiuyu Li, Felix Hohne, and Ser-Nam Lim. New benchmarks for learning on non-homophilous graphs. *arXiv preprint arXiv:2104.01404*, 2021.
- Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.
- Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*, 2021.
- Haitao Mao, Zhikai Chen, Wei Jin, Haoyu Han, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Demystifying structural disparity in graph neural networks: Can one size fit all? *arXiv preprint arXiv:2306.01323*, 2023.
- Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

- Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pp. 4602–4609. AAAI Press, 2019.
- Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampásek. Attending to graph transformers. *arXiv preprint arXiv:2302.04181*, 2023.
- OpenAI. Gpt-4 technical report, 2023.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.
- Ladislav Rampásek, 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.
- Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*, pp. 2535–2546, 2021.
- Jiahui Shi, Vivek Chaurasiya, Yozen Liu, Shubham Vij, Yan Wu, Satya Kanduri, Neil Shah, Peicheng Yu, Nik Srivastava, Lei Shi, et al. Embedding based retrieval in friend recommendation. 2023.
- Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509*, 2020.
- Hamed Shirzad, Ameya Velingker, Balaji Venkatachalam, Danica J Sutherland, and Ali Kemal Sinop. Exphormer: Sparse transformers for graphs. *arXiv preprint arXiv:2303.06147*, 2023.
- Xianfeng Tang, Yozen Liu, Xinran He, Suhang Wang, and Neil Shah. Friend story ranking with edge-contextual local graph convolutions. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 1007–1015, 2022.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.
- Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.
- Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1):396–413, 2020.
- Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.
- Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

- Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.
- 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.
- Qitian Wu, Chenxiao Yang, Wentao Zhao, Yixuan He, David Wipf, and Junchi Yan. Diffformer: Scalable (graph) transformers induced by energy constrained diffusion. *arXiv preprint arXiv:2301.09474*, 2023.
- Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.
- Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.
- 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? In *Advances in Neural Information Processing Systems*, 2021.
- Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.
- Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.
- Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.
- Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less neural networks: Teaching old mlps new tricks via distillation. *arXiv preprint arXiv:2110.08727*, 2021.
- 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.
- Jianan Zhao, Chaozhuo Li, Qianlong Wen, Yiqi Wang, Yuming Liu, Hao Sun, Xing Xie, and Yanfang Ye. Gophormer: Ego-graph transformer for node classification. *arXiv preprint arXiv:2110.13094*, 2021.
- Wenhao Zhu, Tianyu Wen, Guojie Song, Xiaojun Ma, and Liang Wang. Hierarchical transformer for scalable graph learning. *arXiv preprint arXiv:2305.02866*, 2023.

A APPENDIX

A.1 ADDITIONAL RUNTIME ANALYSIS

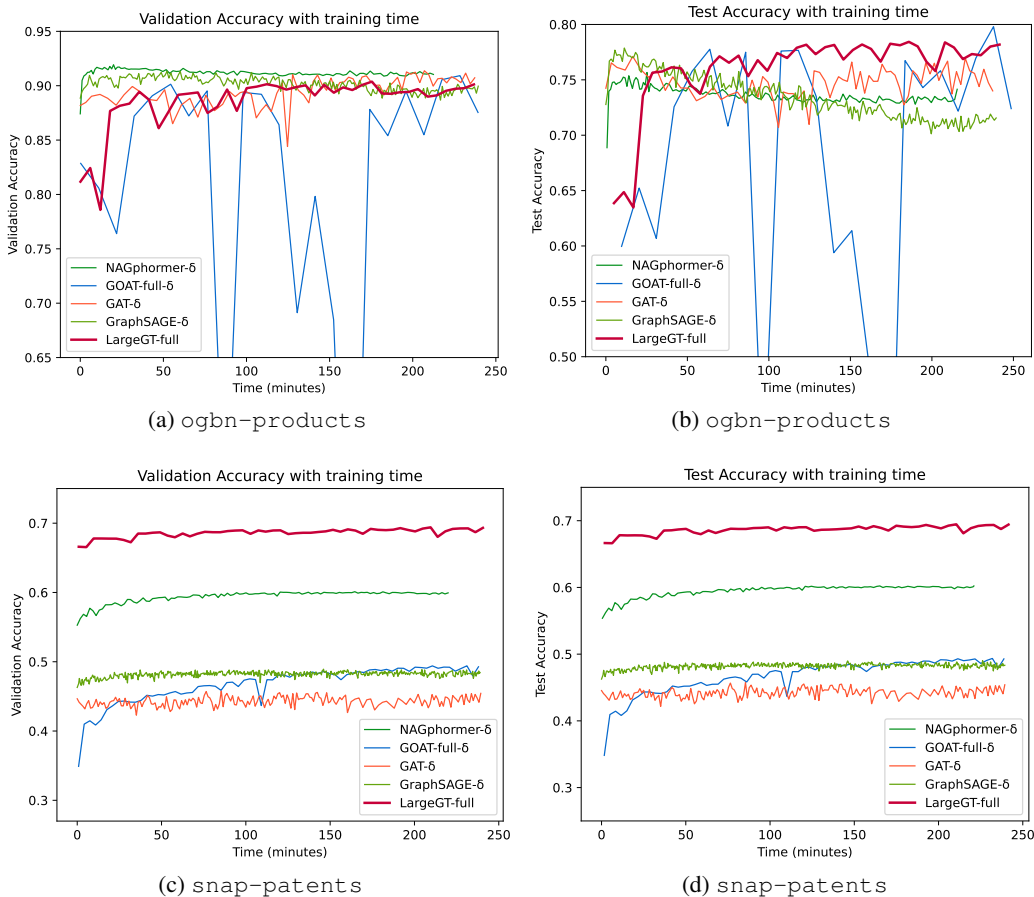


Figure A.1: Validation and test curves of the models with the running time in minutes.

Figure A.1 presents a convergence analysis of the scalable baselines with LargeGT with their validation and test performance for an unlimited number of epochs run within a computation budget of 4 hours (240 minutes). We observe better generalization curves of our proposed architecture against the baselines as the latter are largely affected by NS (neighbor sampling) while our architecture is based on the design principles which addresses some limitations of the NS used during training in the baselines. Additionally, recall from Section 4, we report a $3\times$ faster epoch time of LargeGT vs. GOAT-full- δ on ogbn-products. This is influential in the faster convergence of LargeGT as observed for ogbn-products in Figure A.1, where the generalization curves of LargeGT-full for < 90 minutes is similar to that of GOAT-full- δ for the entire 240 minutes, resulting in an earlier learning stability.

A.2 LOCALMODULE

One of the main contributions of LargeGT (Section 3.2) in building localized representations is the introduction of a tokenization strategy that prepares a fixed set of input tokens for each graph node which is processed in the Transformer encoder of LOCALMODULE to produce local feature representations of the node. For this, first a set of K local nodes are sampled offline using LOCALNODES Algorithm 1, which is then leveraged to prepared $3K$ tokens for each node during the mini batching step, following INPUTTOKENS Algorithm 2. The combination of the sampled nodes and the context features in Algorithm 2 effectively allows a receptive field of up to 4 hops for a node i in consideration, using only 2 hop operations. This is illustrated in Figure A.2. We compute the 1 hop and 2

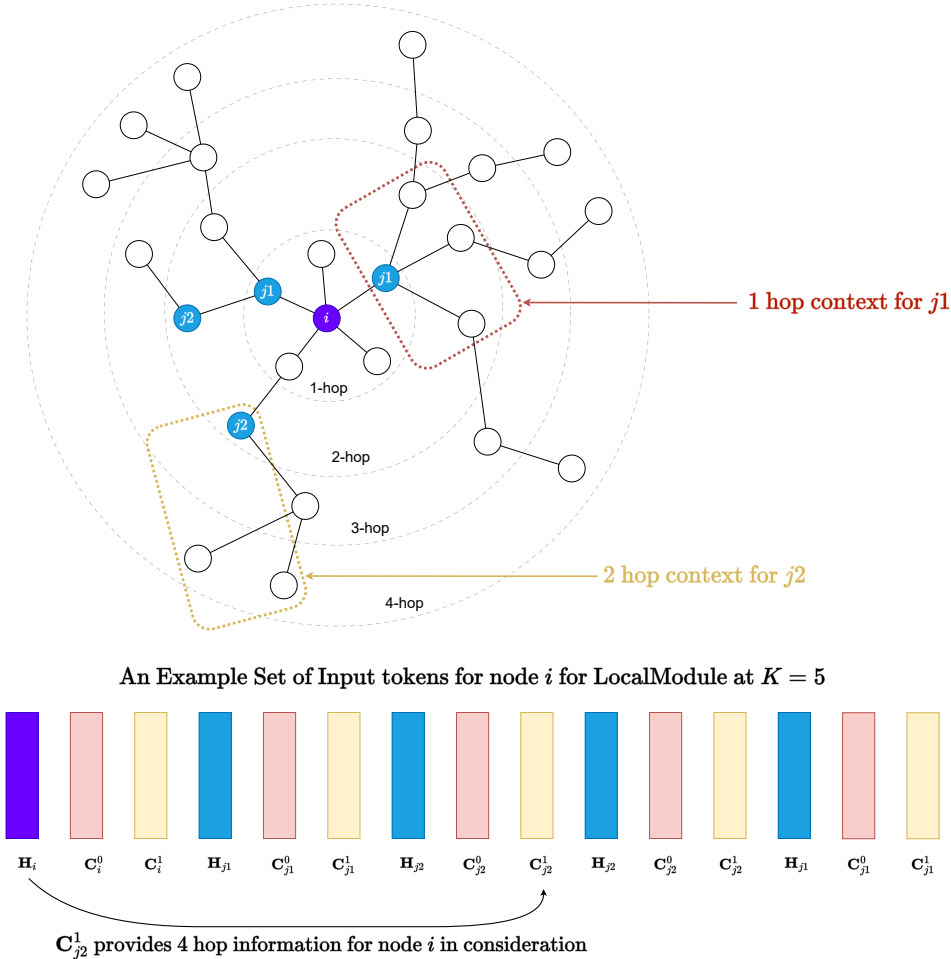


Figure A.2: An example illustration of how the tokenization strategy using LOCALNODES and INPUTTOKENS proposed in LargeGT which is input into LOCALMODULE allows an effective receptive field of up to 4-hops for a node i in consideration.

hop context features for all nodes using $\mathbf{C}^0 = \tilde{\mathbf{A}}\mathbf{H}$ and $\mathbf{C}^1 = \tilde{\mathbf{A}}^2\mathbf{H}$, respectively, where $\tilde{\mathbf{A}}$ is the normalized adjacency matrix and \mathbf{H} is the node feature matrix.

In the figure, where K is set to 5 for simple illustration, we can observe that there are $K - 1 = 4$ sampled nodes for the node i , where the nodes sampled from 1 hop are represented as $j1$ and from 2 hop are represented as $j2$. The 2-hop context feature for $j2$ will also form part of the token set for the node i , following INPUTTOKENS Algorithm 2. As such, the set of tokens of size $3K$ will include vectors which represent the a node i 's 4 hop neighborhood information.

A.3 GLOBALMODULE

The algorithm GLOBALMODULE in LargeGT is adapted from Kong et al. (2023) which uses a single layer Transformer encoder to allow every node in a mini-batch to global graph nodes, *conceptually* through a fixed sized B centroids. The B centroid tokens are stored in a codebook which is defined as $\boldsymbol{\mu} \in \mathbb{R}^{B \times D}$ and are updated using an Exponential Moving Average (EMA) K-Means algorithm (Kong et al., 2023) at every mini-batch step. In Algorithm 3, MLP_a and MLP_b are two multi-layer perceptron modules with separate learnable parameters, while $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ are the learnable parameters of the Transformer encoder.

Algorithm 3 GLOBALMODULE: Algorithm to output global representations of nodes in mini-batch

Require: Hidden features $\mathbf{H}_{\text{in}} \in \mathbb{R}^{M \times D}$ for M nodes in a mini-batch, $\boldsymbol{\mu}$ is the centroid computed by the K-Means algorithm, \mathbf{P} is the centroid assignment index for each node.

Ensure: Return the output representations of all nodes in the mini-batch $\mathbf{H}^{\text{global}}$.

```

1: for  $i = 0$  to  $M - 1$  do
2:    $\mathbf{x} \leftarrow \text{MLP}_a(\mathbf{H}_i^{\text{in}})$ 
3:    $\mathbf{q} \leftarrow \mathbf{x}\mathbf{W}_Q$ 
4:    $\mathbf{K} \leftarrow \boldsymbol{\mu}\mathbf{W}_K$ 
5:    $\mathbf{V} \leftarrow \boldsymbol{\mu}\mathbf{W}_V$ 
6:    $\hat{\mathbf{H}}_i^{\text{global}} \leftarrow \text{Softmax}\left(\frac{\mathbf{q}\mathbf{K}^\top}{\sqrt{D}} + \log(\mathbf{1}_n\mathbf{P})\right)\mathbf{V}$ 
7:    $\mathbf{H}_i^{\text{global}} \leftarrow \text{MLP}_b(\hat{\mathbf{H}}_i^{\text{global}})$ 
8:   Update  $\boldsymbol{\mu}$  using  $\mathbf{x}$  through Exponential Moving Average (EMA) K-Means algorithm.
9: end for

```

A.4 EXTENDED BASELINE RESULTS

As mentioned in Section 4.1, we compare LargeGT with ‘Scalable Baselines’ which are constrained versions of existing MPNNs or GT baselines, denoted by Model- δ . The decision to compare LargeGT with 2-hop constrained versions of the baselines was to maintain a consistent scope of expensive computation incurred across all models. For instance, the cost of fetching l -hop neighbors for a node in a very large graph would be the same and agnostic of any selected model. Nevertheless, in this section, we provide an extended comparison of LargeGT with both ‘Scalable Baselines’ as well as their original models, for `ogbn-products`. Note that for the original models, we adopt hyperparameters in their respective papers. From Table A.1, we can observe that the best performing model remains the same while LargeGT, which only use up to 2-hop operations, remains better and competitive compared to the best performing model. Additionally, it is obvious that the ‘Original Baselines’ that employ greater than 2-hop operations are computationally demanding (upto $4\times$ in some cases), as compared to the ‘Scalable Baselines’.

Table A.1: Extended results for `ogbn-products` where LargeGT is compared to Model- δ as well as the original baselines without the $-\delta$ constraint.

Model	Original Baselines		Scalable Baselines (- δ)	
	Test Acc	Epoch Time	Test Acc	Epoch Time
GraphSAGE	78.17	22s	76.62	7s
GAT	79.21	86s	77.38	31s
GT-sparse	67.87	40s	60.76	12s
NAGphormer	77.71	22s	75.28	8s
GOAT-local	81.29	490s	81.17	120s
GOAT-global	70.28	3.5s	70.28	3.5s
GOAT-full	81.21	500s	79.88	205s
LargeGT-local		–	78.95	45s
LargeGT-full		–	79.81	68s

A.5 EXPERIMENTAL DETAILS

In our experiments, we use the baselines GraphSAGE (Hamilton et al., 2017), GAT Veličković et al. (2018), GT-sparse (Dwivedi & Bresson, 2021; Shi et al., 2020), NAGphormer (Chen et al., 2022) and GOAT (Kong et al., 2023) to compare with our proposed architecture LargeGT. We follow the setup of the respective codebase of NAGphormer³ and GOAT⁴ to conduct the baseline experiments, while

³<https://github.com/JHL-HUST/NAGphormer>

⁴<https://github.com/devnkong/GOAT>

for the remaining, we follow their OGB implementation⁵. The hardware setup and hyperparameters of the experiments are as follows.

A.5.1 HARDWARE

The experiments on `ogbn-products` and `snap-patents` are conducted on Tesla-V100 GPU with 16GB GPU memory, 24 CPU cores and 156GB RAM, while that of `ogbn-papers100M` are on A40 GPU with 48GB GPU memory, 128 CPU cores and 1024GB RAM.

A.5.2 HYPERPARAMETERS

Table A.2: Hyperparameters for `ogbn-products`

	GraphSAGE- δ	GAT- δ	GT-sparse- δ	NAGphormer- δ	GOAT- δ	LargeGT
batch size	1024	512	1024	200	512	1024
hidden dim	256	128	128	512	256	256
heads	–	4	4	4	2	2
pos enc	–	–	–	–	node2vec (64)	node2vec (64)
centroids (B)	–	–	–	–	4096	4096
NS	[20, 10]	[20, 10]	[20, 10]	–	[20, 10]	–
dropout	0.5	0.5	0.3	0.3	0.5	0.5
learning rate	3e-3	1e-3	1e-3	–	1e-3	1e-3
weight decay	0.0	0.0	0.0	1e-5	0.0	0.0
warmup steps	–	–	–	1000	–	–
peak learning rate	–	–	–	1e-3	–	–
end learning rate	–	–	–	1e-4	–	–
learning patience	–	–	–	30	–	–
hops	–	–	–	2	–	–

⁵<https://github.com/snap-stanford/ogb>

Table A.3: Hyperparameters for snap-patents

	GraphSAGE- δ	GAT- δ	GT-sparse- δ	NAGphormer- δ	GOAT- δ	LargeGT
batch size	1024	512	1024	200	2048	2048
hidden dim	256	128	128	512	128	128
heads	–	4	4	4	2	2
pos enc	–	–	–	–	node2vec (64)	node2vec (64)
centroids (B)	–	–	–	–	4096	4096
NS	[20, 10]	[20, 10]	[20, 10]	–	[20, 10]	–
dropout	0.5	0.5	0.3	0.2	0.5	0.5
learning rate	3e-3	1e-3	1e-3	–	1e-3	1e-3
weight decay	0.0	0.0	0.0	1e-5	0.0	0.0
warmup steps	–	–	–	1000	–	–
peak learning rate	–	–	–	1e-3	–	–
end learning rate	–	–	–	1e-4	–	–
learning patience	–	–	–	30	–	–
hops	–	–	–	2	–	–

Table A.4: Hyperparameters for ogbn-papers100M

	GOAT- δ	LargeGT
batch size	2048	1024
hidden dim	768	512
heads	2	2
pos enc	node2vec (128)	node2vec (128)
centroids (B)	4096	4096
NS	[20, 10]	–
dropout	0.5	0.5
learning rate	1e-3	1e-3
weight decay	0.0	0.0