TorchGT: A Holistic System for Large-Scale Graph Transformer Training

Meng Zhang*[1,3] Jie Sun*[4] Qinghao Hu[1,3] Peng Sun[3,5] Zeke Wang[4] Yonggang Wen[2]
Tianwei Zhang☆[2]

[1]S-Lab, Nanyang Technological University [2]NTU [3]Shanghai AI Laboratory
[4]Zhejiang University [5]SenseTime Research
{meng.zhang, qinghao.hu, ygwen, tianwei.zhang}@ntu.edu.sg
sunpeng1@sensetime.com *{jiesun, wangzeke}@zju.edu.cn*

***Abstract*—Graph Transformer is a new architecture that surpasses GNNs in graph learning. While there emerge inspiring algorithm advancements, their practical adoption is still limited, particularly on real-world graphs involving up to millions of nodes. We observe existing graph transformers fail on large-scale graphs mainly due to heavy computation, limited scalability and inferior model quality.**

Motivated by these observations, we propose TorchGT, the *first* efficient, scalable, and accurate graph transformer training system. TorchGT optimizes training at three different levels. At *algorithm* level, by harnessing the graph sparsity, TorchGT introduces a Dual-interleaved Attention which is computation-efficient and accuracy-maintained. At *runtime* level, TorchGT scales training across workers with a communication-light Cluster-aware Graph Parallelism. At *kernel* level, an Elastic Computation Reformation further optimizes the computation by reducing memory access latency in a dynamic way. Extensive experiments demonstrate that TorchGT boosts training by up to 62.7× and supports graph sequence lengths of up to 1M.

***Index Terms*—Graph Transformer, Distributed Training, Sparse Attention, Graph Parallelism**

I. Introduction

Graph-structured data has long been prevalent and indispensable in many real-life applications such as social network construction and molecule analysis. Thus, there emerges a specific family of graph learning methods, namely graph neural networks (GNNs) [1]–[3]. GNNs have gained giant breakthroughs and exhibit impressive performance in many tasks such as node classification [4]–[7] and link prediction [3], mainly due to their message passing mechanism [8], which models the inherent properties of graph structures. However, this module in classic GNNs also leads to commonly acknowledged over-smoothing [9], over-squashing [10], [11] and limited expressivity [12] issues.

To address these deficiencies, a latest approach called *graph transformer* shows more promising power in capturing the inter-dependencies among nodes. Graph transformer is built upon the classical Transformer [13] which allows nodes to attend to all other nodes, and integrates multiple graph structure encodings to include important graph properties. Due to the great modeling capability, graph transformer has garnered

*Equal Contribution. ☆Corresponding author.

TABLE I: Graph transformers outperform classical GNNs on graph-level and node-level (Flickr) tasks.

Model		ZINC Test MAE↓	PCQM4M-LSC Validate MAE↓	Flickr Test Acc.(%)↑
Traditional GNNs	GAT	0.384	-	54.29
	GCN	0.367	0.169	61.49
Graph Transformers	GT	0.226	0.141	**68.59**
	Graphormer	**0.122**	**0.123**	66.16

surging interest in recent years and a large number of models have been proposed [14]–[17]. Existing graph transformers mainly operate by treating graph nodes as input tokens and constructing an input sequence consisting of all the graph nodes. Besides, graph structure encoders are designed as a graph adaptation of the original Transformer architecture. By integrating structural information, graph transformers exhibit competitive performance and outperform traditional message-passing GNNs (e.g., GCN [1] and GAT [3]) on both node classification [16]–[21] and graph classification [14], [15], [22] tasks, as shown by Table I. We can obviously see graph transformers obtain the highest scores than GNNs on all tasks.

Real-world graphs can easily involve millions of nodes [23], [24], making the sequence length enormously large. For example, in the current graph transformers' operation way, processing the citation graph dataset ogbn-papers100M from Open Graph Benchmark [23] (including more than 100 million nodes) requires high dimensional inputs with prohibited sequences. Moreover, as illustrated by the profiled results in §II-B, training graph transformers with long sequence is crucial for model quality and the development of versatile graph transformer application scenarios. However, we find most existing graph transformer research works [14], [15], [25], [26] are only limited to small graphs due to a lack of compatible systems tailored for the graph transformer model training with long sequences. More specifically, there are three deficiencies in current works:

First, graph transformers with standard attention have poor scalability to long sequences, due to the computation and memory complexity of $O(N^2)$, quadratic on the number of nodes (N) in a graph [14], [15], [27]–[30]. Taking fully-connected attention as graph foundation encoders captures the implicit all-pair influence beyond neighboring nodes, but also

SC24, November 17-22, 2024, Atlanta, Georgia, USA
979-8-3503-5291-7/24/$31.00 ©2024 IEEE

limits existing graph transformers only on small-graph applications [14], [15], [27]–[30]. Figure 2 demonstrates that even with a state-of-the-art attention library, i.e., FlashAttention [31], the computation of the dense attention mechanism is still a bottleneck during the graph transformer training.

Second, current algorithms either compromise model quality or are only applicable to a single graph learning task. To reduce computation pressure, some graph transformers shorten the input sequences by harnessing neighbor sampling [16], [32] similar adopted in classic GNNs [2], [33], [34]. Others like [25] attempt to overcome the quadratic complexity by replacing full attention with approximate attention methods. However, using sampling methods or simply adopting sparse patterns like [35], [36] loses critical connectivity information and thus sacrifices model precision. On the other hand, some works [17], [20], [21] use self-defined adapted attention modules to reduce memory consumption, but are limited to a specific task, e.g., node classification. They are neither general to versatile graph learning tasks nor portable to be scaled in large-scale training.

Third, no existing works exploit systematic optimizations to realize efficient and scalable training. Several graph transformers [25], [26] apply graph structure to relieve the computation burden. However, this sparse pattern is highly irregular in memory access due to the skewed property of graphs, which is challenging for optimizing the system throughput. Moreover, with large datasets, the memory consumption of model activations grows rapidly, necessitating a scalable system design and memory optimization. But existing works [14]–[16], [25], [26] only focus on the implementation of graph transformers in a single GPU, thus limited to very small graphs. Although there has been a breakthrough for large language models (LLMs) by partitioning along the input sequence dimension and training long sequences across devices [37]–[40], those sequence parallelism ways cannot be directly transplanted on graph transformers due to the extra graph encodings and neglection of structure properties.

To bridge these gaps, we design TORCHGT, the ***first*** distributed training system that scales *graph transformer model* to large graphs with billions of edges. Our system abides four design goals: ***scalable***, ***efficient***, ***convergence-maintained***, and ***task-agnostic***. Existing graph transformer works neither facilitate efficiency by well-designed parallelism from the system perspective nor propose scalable algorithms for universal graph learning tasks, thus making it challenging to meet those goals. This hinders the practical development of advanced graph transformer models on real-world graphs. The core design of TORCHGT derives from the following three key insights. First, *the learning of graph transformers highly benefits from graph structures.* Specifically, the structure of many real-world graphs is highly sparse [41]–[43], which reflects the inherent vertex-vertex interactions. This sparsity could be a guide for how graph transformers attend to nodes to reduce computation costs while maintaining correct connections. In addition, considering the structural property in the system design also contributes to optimal hardware throughput. Second, *the order of input graph tokens is alterable.* Unlike inputs in famous LLMs like GPT [44] whose token order is crucial for model understanding and generation process [44], [45], graph transformers focus more on connections between nodes. Thus, we can modify the input arrangement to exploit graph properties (e.g., local clusters) for more specialized optimizations. Third, *the block-sparse format is a good match for irregular graph clusters.* Block-sparse formats store data contiguously in memory, reducing storage overheads and memory access. But directly exploiting it on dense attention matrices will drop connectivity and result in substantial accuracy loss [46], [47]. However, by integrating it into our specialized clustered pattern, we find the computation can be further accelerated while maintaining model accuracy.

As such, our key idea is to design an accuracy-maintained and compute-efficient system from both algorithm and system perspectives to support large-scale graph transformer training. Specifically, TORCHGT consists of three key innovations. **Dual-interleaved Attention** is a local-global interleaved attention that integrates graph structural topology into the attention module and selectively combines the global information into the attention with the graph structure search, which efficiently speeds up the attention computation while maintaining the models' qualities. **Cluster-aware Graph Parallelism** splits the input graph tokens according to the cluster nature of graphs, thus boosting the attention computation throughput and facilitating system scalability. It also allows us to take advantage of the cluster feature in more fine-grained kernel optimizations. Inspired by the block-sparse format, **Elastic Computation Reformation** dynamically transfers the clustered attention pattern into a specialized cluster-sparse format to reduce the irregular memory access latency. It includes an *Auto Tuner* to automatically control the transfer to maintain the model convergence. Through extensive experiments, we show TORCHGT successfully achieves scalable and efficient graph transformer training on large graphs. It also boosts training by up to 62.7× across various graph learning tasks while maintaining accuracy.

In summary, we make the following contributions:

★ TORCHGT is the *first* graph transformer system that facilitates efficient, scalable, and accurate training on large-scale graphs as well as universal graph learning tasks.

★ TORCHGT is the *first* to identify the major challenges that hinder existing graph transformers from scaling to large graphs and explore the graph-specific optimization opportunities which are neglected previously.

★ We propose three key techniques to meet all design goals from algorithm and system co-design perspectives.

★ Experiments show TORCHGT achieves up to 62.7× speedup and near-linear scalability, supporting graph sequence lengths of up to millions.

II. BACKGROUND AND MOTIVATION

A. Graph Transformer

Graph transformer architecture has attracted surging attention in graph representation learning in recent years [48].

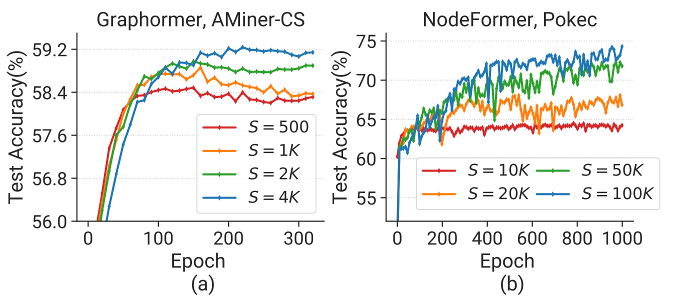

Fig. 1: The test accuracy of graph transformers when trained with different sequence lengths S.

Current representative graph transformers integrate graph structural encodings into the input and attention map in the Transformer architecture. The input sequence is built by tokens generated with graph attributes. Specifically, some works [14]–[16], [18], [49]–[51] calculate node positional encodings beforehand and add them to the inputs before the attention module. Other works [14], [15], [17], [32], [52] add graph structural information into the self-attention matrix as bias. Several works [19], [25], [26] combine message-passing GNNs and the attention mechanism together. Here we only focus on the former two types of graph transformers since they are currently most representative.

A basic Transformer consists of multi-head attention (MHA) and feed-forward network (FFN) which contains two linear layers. Given an input sequence $\boldsymbol{H} = [h_1, \cdots, h_S]^\top \in \mathbb{R}^{S\times d}$ where S is the sequence length and d is the hidden dimension, MHA first projects its input $\boldsymbol{H}$ to three subspaces: $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ with projection weight matrices $\boldsymbol{W_Q} \in \mathbb{R}^{d\times d_{\boldsymbol{K}}}, \boldsymbol{W_K} \in \mathbb{R}^{d\times d_{\boldsymbol{K}}}, \boldsymbol{W_V} \in \mathbb{R}^{d\times d_{\boldsymbol{V}}}$. The MHA output is calculated as:

$$\boldsymbol{H'} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_{\boldsymbol{K}}}}\right)\boldsymbol{V} \tag{1}$$

where $d_{\boldsymbol{K}}$ is the second dimension of $\boldsymbol{K}$. MHA captures the pair-wise similarity of input tokens in the sequence.

For a graph $G = (V, E)$ with nodes $V = \{v_1, \cdots, v_N\}$ and edges $E = \{e_1, \cdots, e_{|E|}\}$, here we list the formulation of Graphormer [14] as an example:

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)} \tag{2}$$

$$\boldsymbol{A}_{ij} = \frac{(h_i\boldsymbol{W_Q})(h_j\boldsymbol{W_K})^\top}{\sqrt{d_{\boldsymbol{K}}}} + bias_{\phi(v_i,v_j)} \tag{3}$$

where $h_i^{(0)}$ is the beginning attribute of node i, x_i is the node feature, and $z^-, z^+ \in \mathbb{R}^d$ are learnable embeddings specified by the in-degree $\deg^-(v_i)$ and out-degree $\deg^+(v_i)$. The encodings in Equation 2 allow the attention to capture the node importance. $\boldsymbol{A}_{ij}$ is the (i, j)-element of Query-Key product matrix, namely the attention coefficient. $bias_\phi$ is a learnable scalar shared across all layers, and $\phi(v_i, v_j)$ is the distance of the shortest path (SPD) between node v_i and v_j, which is the shortest hops that v_i needs to pass to reach v_j.

B. Long Sequence for Graph Transformers

For better illustration, we categorize current graph learning tasks into two types to discuss the need of training in long sequences: (1) graph-level task, and (2) node-level task.

Long Sequence for Graph-level Tasks. For such tasks, the input sequences represent a set of graphs while the output is a set of labels representing the types of corresponding graphs. When processed by graph transformers, all nodes in each input graph need to be encoded as input tokens and are concatenated into an input sequence. As such, the length of each sequence equals the number of nodes in each graph. In this task, if the graph size, i.e., the number of nodes, grows very large, the input sequence can be too long to be trained by current methods. For instance, the MalNet [53] dataset contains graphs with up to 552K nodes.

Long Sequence for Node-level Tasks. These tasks classify each node in an input graph with a specific label. In the node classification task, the input sequences can either encode all nodes in the graph or a mini-batch of nodes. For the former case, the input sequence can be enormously long for large-scale graphs, which is not supported by most models. For the latter, with a larger batch size, both the training throughput and the trained model quality can be improved. Both cases validate the necessity and advantages of long sequence training.

However, existing graph transformers have some inherent constraints in performing the above tasks. While graph-level scenario has been explored in [14], [15], existing endeavors do not generalize to large-scale graphs endemic to node-level prediction. Our TorchGT strives to include both tasks by joint algorithm-system design. The scale of graphs applicable to current models is still limited, thus leaving long sequence training still an urgent necessity. Besides, training large-scale graphs in short sequence suffers from lower training throughput, downgraded model quality and limited graph transformer applications. Figure 1 illustrates the impact of sequence length on the test accuracy of two representative models Graphormer [14] and NodeFormer [17] on two datasets. Both models show superior performance on longer sequences. On the AMiner-CS dataset, Graphormer with a 4K sequence length improves the test accuracy by up to 0.9% compared to the short sequences. On the Pokec dataset, sampling-based NodeFormer with 100K sequence length outperforms the case with 10K sequence length by a staggering 12% accuracy. These results necessitate the need for long sequence training of graph transformers.

C. Issues and Opportunities

Most existing graph transformer works [14], [15], [19], [25], [26], [32] are only limited on small graphs due to a lack of compatible systems tailored for the graph transformer model training with long sequences. They have three main issues when applied to long sequence training.

I1: Attention Computation Bottleneck. Graph transformers with standard (dense) attention treat the graph as fully-connected with the MHA mechanism calculating attention for all node pairs. Thus, it requires the computation complexity of the attention module to be quadratic on the number of

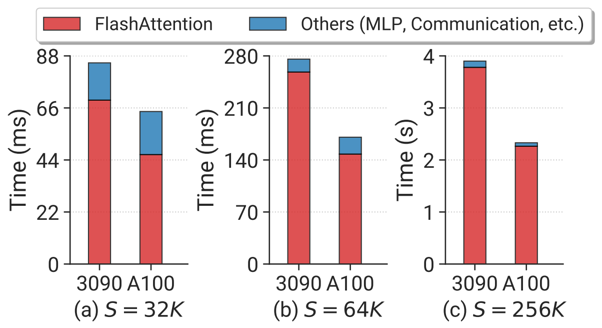

Fig. 2: Training iteration time breakdown when training Graphormer on ogbn-products in different sequence lengths on two types of GPUs: RTX 3090 and A100.

nodes (N^2) in a graph, which limits the models' scalability to extremely long sequences. Currently, there is a breakthrough in standard attention optimization, i.e., FlashAttention [31]. FlashAttention accelerates the attention module by fusing the IO-bound GPU kernels like `Softmax` and `Dropout` within the attention computation. However, even with FlashAttention to train graph transformers with long sequences, e.g., sequence length of 512K, we still identify that the attention module dominates the overall training time.

To show this, we conduct an experiment to record the iteration time breakdown when using FlashAttention, as illustrated in Figure 2. Current FlashAttention does not support the modified attention module like those augmented with bias encodings [14], [15], [17], [32], so we disable the bias in this experiment to only examine the computation efficiency. We separate the computation time of FlashAttention from the comprehensive training iteration. We can obviously observe that no matter on longer or shorter sequences, attention computation still dominates over 80% of training time, indicating a severe attention bottleneck. However, both the standard attention and FlashAttention fail to leverage one important characteristic of the graph, namely its topology structure, which we find profoundly impacts the effectiveness of system optimizations.

I2: Degraded Model Convergence and Limited Tasks. Many efforts [17], [25], [35], [54] have been made to overcome the computation bottleneck of the attention module. Among them, [54] prunes the attention module and leaves a major backbone to reduce the computation cost for LLMs. Some works [35], [36], [55], [56] propose sparse patterns on attention to scale linearly, but most of them are designed for natural language processing (NLP). They cannot be simply grafted to graph transformers since they fail to consider the inherent graph structure information when approximating attention, thus resulting in subpar model performance. Several graph transformers [16], [32], [57] harness neighbor sampling or graph pooling that only selects a subset of nodes to be trained at each iteration, without reducing the computation complexity. Nonetheless, all the above methods sacrifice model precision by dropping the connectivity information.

In the graph domain, efficient attention is not well studied. Few graph transformers like [26] apply the graph structure to attend nodes and maintain graph-specific information. However, they limit the implementation to the GNN-encoding-based model architecture, e.g., GraphGPS [25], and highly rely on the message-passing scheme for excellent model performance. Other methods [17], [20], [21] use self-defined adapted attention modules to achieve linear complexity. However, all those works are constrained to a single application task, failing to generalize to versatile graph tasks. Additionally, with GNN structure encodings or self-defined attention, the model can hardly be scaled to multiple workers.

TABLE II: Backward (BW) time of topology-pattern & dense counterpart when training Graphormer on ogbn-products.

Seq. Length	S=64K	S=128K	S=256K	S=512K
Topology-pattern BW. Time/ms	116.99	234.28	499.289	963.91
Dense BW. Time/ms	1.53	3.78	10.02	29.01

I3: Lack of Specific System Optimizations. As far as we know, currently there is no existing framework to optimize graph transformer training from the system level. FFN operations in MHA are dense in computation and regular in memory access. However, utilizing graphs on the attention module is sparse in computation and requires irregular and fine-grained memory access due to the skewed nature of graph structures, which inevitably becomes the performance barrier.

Existing solutions [15], [25], [26] directly apply graph topology in the attention computation while ignoring the pattern differences between graph transformer and standard Transformer-based models. To better illustrate, we experimentally examine the impact of irregular memory access by the topology-pattern attention in Table II. The topology-induced memory access latency is tremendous, reaching up to 33.2× slowdown than dense computation. To increase models' scalability, recent works [37]–[40] split the input sequences and distribute the computation across devices. However, this parallelism neglects various graph encoding modules and fails to distinguish input tokens in graph domain from tokens in NLP. Those differences invoke specialized and dedicated system designs for graph transformers towards more efficient memory optimizations and more aggressive parallelism.

III. TorchGT Design

We propose TorchGT, an algorithm-system co-optimized system tailored for graph transformer training on large-scale graphs. It follows four design principles:

- *Scalable*. TorchGT can scale graph transformer training to extremely large graphs (solving **I3**).
- *Efficient*. TorchGT reduces over 90% computation required by standard attention, overcoming the attention computation bottleneck (solving **I1**).
- *Convergence-maintained*. TorchGT maintains comparable model convergence to the graph transformer with standard attention, and successfully balances the trade-off between training efficiency and model quality (solving **I2**).
- *Task-agnostic*. TorchGT generalizes to various graph transformer models and graph learning tasks (graph-level and node-level) (solving **I2**).

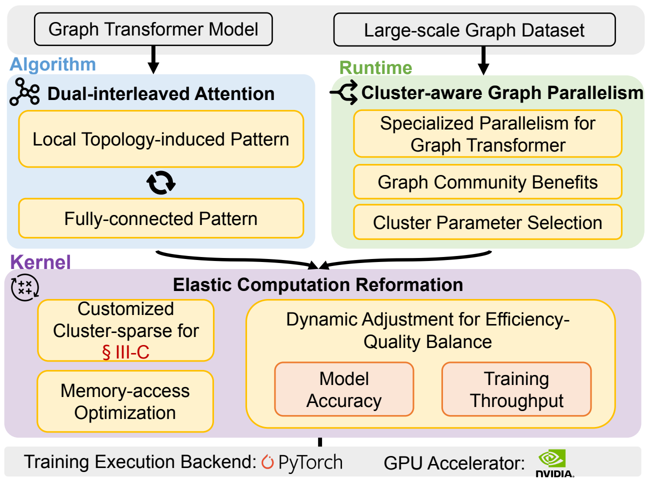
Graph Transformer Model
Large-scale Graph Dataset
Algorithm
Dual-interleaved Attention
Local Topology-induced Pattern
Fully-connected Pattern
Runtime
Cluster-aware Graph Parallelism
Specialized Parallelism for Graph Transformer
Graph Community Benefits
Cluster Parameter Selection
Kernel
Elastic Computation Reformation
Customized Cluster-sparse for § III-C
Memory-access Optimization
Dynamic Adjustment for Efficiency-Quality Balance
Model Accuracy
Training Throughput
Training Execution Backend: PyTorch
GPU Accelerator:

Fig. 3: Overview of TORCHGT architecture and workflow.

A. System Overview

Motivated by all the observations in §II-C, our key idea is to co-design an accuracy-maintained and compute-efficient attention module with a graph-parallelism-enabled system framework to support long sequence training. As shown in Figure 3, TORCHGT intelligently optimizes training across three levels from the top to bottom hierarchy: *algorithm*, *runtime* and *kernel*. We propose a topology-induced and accurate attention algorithm in the algorithm level. We present a novel cluster-aware graph parallelism to scale the training in the runtime level. In the kernel level, we design a memory-optimized computation pattern specialized for clustered attention. Specifically, TORCHGT consists of three key modules:

- *Dual-interleaved Attention*: In the algorithm level, it integrates locally graph-induced topology into the attention computation pattern and periodically overlays it with the fully-connected information, which efficiently reduces the computation burden while maintaining the model's quality as much as possible. It is tailored for versatile graph transformer models with local-global interleaved attention.
- *Cluster-aware Graph Parallelism*: From the distributed runtime perspective, we design a cluster-aware graph parallelism tailored for graph transformers. It splits the graph tokens in sequences according to the clustering nature of graphs, thus computing attention with locality and facilitating the system scalability.
- *Elastic Computation Reformation*: It reformats the graph-induced pattern obtained at the runtime level into our customized and fine-grained cluster-sparse pattern at the underlying execution kernel level. It further improves the attention computation throughput by greatly alleviating irregular memory access. To balance the efficiency-quality trade-off, we build an *Auto Tuner* to make an elastic transfer of cluster sparsity.

B. Dual-interleaved Attention

Motivated by **I1** in §II-C, TORCHGT explores the opportunity of integrating graph structure to reduce the substantial computation cost. For optimizations aiming at graph transformers, we design an interleaved attention to realize a local-global interleaved attention and ensure model convergence.

Local Topology-induced Pattern. In NLP tasks, the tokens in a sequence represent words, while in graph transformers the tokens are nodes of the input graph. Besides, most graph transformers like [14], [17] adopt the standard attention, which can be viewed as a fully-connected graph since all tokens attend to every other token to perform inner products, leading to quadratic complexity. Motivated by the sparse attention methods [35], [36], we find the local topology-induced pattern that makes use of the underlying structure of the input graph is desirable to guide the pair-wise node interactions. Graphs innately own two desiderata for attention mechanism: (1) *small pair-wise node interactions (large sparsity)*, and (2) *data locality*. In addition, most sparse patterns in NLP are only approximations [36] to their dense counterparts under specific contexts, while in our scenario the graph structure is real and valid, without the need of approximations. Thus, we compute attention by attending each node to its immediate neighbors in the graph, reducing the interacted node pairs.

We formulate the local topology-induced pattern as below. To train on a graph $G = (V, E)$, we generate an input sequence $\boldsymbol{S} \in \mathbb{R}^{S\times d}$ comprised of graph tokens corresponding to a node set $\tilde{V} \in V$. $\tilde{V}$ can be equal to either the whole nodes V, e.g., in graph-level tasks, or a subset of V, e.g., in node-level tasks if the node number is too large. For each node set $\tilde{V}$, we construct a local attention graph $\tilde{G} = (\tilde{V}, \tilde{E})$, where the edge set $\tilde{E}$ is also a subset of the original edge set E. If there exists a global token in the model that attends to all nodes in the graph and is attended to by all nodes, we augment $\tilde{E}$ with the global token's edges. The general attention coefficient $\tilde{\boldsymbol{A}}_{ij}$ of graph transformers without graph encodings is computed in: $\tilde{\boldsymbol{A}}_{ij} = \text{softmax}\left(\frac{(h_i \boldsymbol{W}_{\tilde{Q}})\cdot(h_j \boldsymbol{W}_{\tilde{K}})^\top}{\sqrt{d_{\tilde{K}}}}\right)$. The updated node attribute h_i' for each node i is computed as the weighted sum of the features of its neighboring nodes from $\tilde{V}$: $h_i' = \sum_{j\in\mathcal{N}(i)} \tilde{\boldsymbol{A}}_{ij}(h_j \boldsymbol{W}_{\tilde{V}})$, where $\mathcal{N}(i)$ denotes the set of neighboring nodes of node i. Each neighbor's feature vector h_j is weighted by the attention coefficient $\tilde{\boldsymbol{A}}_{ij}$, and these weighted features are summed to update the attribute of node i. By using our local topology-induced pattern $\tilde{G}$, the attention only computes coefficients of connected node pairs.

Interleave Fully-connected Pattern. Implementing the attention computation via the graph structure can greatly reduce the computation cost. However, it sometimes slows the model accuracy and convergence, which can be shown by experiment results in Figure 10. The local graph-induced attention slightly degrades the model convergence, which is mainly because the topology-induced pattern restricts the attention mechanism from extracting the high-order neighboring information. Intuitively, larger sparsity induces more absence in the attention computation, and increases the model error. Building on this, we empirically interleave a fully-connected attention on the local graph-induced attention.

To fill the performance gap between sparse attention and its dense counterpart, we need to figure out when to interleave the dense pattern. Motivated by the sparse attention theories in [58], we conclude three critical conditions under which we use the topology-induced pattern on attention:

- **C1:** Every node in the sequence $\boldsymbol{S}$ always attends to itself.
- **C2:** There exists a Hamiltonian path that directly connects all nodes $\tilde{V}$ in the sequence.
- **C3:** All nodes in the sequence should be able to attend to other nodes, either directly or indirectly after L graph transformer attention layers.

The Hamiltonian path [59] or traceable path is a path in a graph that visits each node exactly once. For each graph $\tilde{G}$ corresponding to the input sequence, the *Dual-interleaved Attention* module searches it across the above three conditions. We use a heuristic approach Dirac's theorem [60] to do quick checks so the overhead is negligible in epoch time. If it satisfies these conditions, we perform attention computation with the topology-induced sparse pattern. Otherwise, TorchGT heuristically determines the current sparse pattern may introduce more errors and we utilize the fully-connected attention mechanism in this case to ensure model quality.

Computation & Memory Complexity. The topology of many real-world graphs can be immensely sparse. For instance, the ogbn-arxiv graph has 169K nodes and 1.2M edges, resulting in a sparsity of 4.1×10^{-5} (the proportion of nonzero elements in the whole adjacency matrix). As a result, the local topology-induced attention significantly reduces the computation and memory-access complexity from $O(N^2)$ to $O(\tilde{E})$. Though we interleave several fully-connected attention occasionally, the overall computation efficiency is still improved significantly.

Model Convergence. Graph-centric attention [26] and classical GNNs [2], [3] prove that sparse attention can maintain the model convergence comparable to its dense counterpart. [18], [58] propose sparse attention can obtain similar universality as dense attention under some assumptions. Borrowing it to TorchGT, our *Dual-interleaved Attention* can provide universal approximation properties that every continuous function f can be approximated to any desired accuracy using a suitable sparse pattern under the three conditions, thus obtaining convergence similar to dense counterparts.

C. Cluster-aware Graph Parallelism

To better fit the topology-induced attention pattern and increase the system scalability, we introduce a graph transformer-specialized parallel training style: *Cluster-aware Graph Parallelism*, which exploits the graph cluster characteristics to guide the distributed training.

Utilization of Graph Cluster. Graph cluster (community) [61], [62] is one essential characteristic of real-world graphs, referring to a subset of nodes within a graph that exhibit a higher degree of connectivity with each other compared to nodes in other parts of the graph. Although the graph structure-based attention in §III-B greatly reduces the computation, this sparse and highly-skewed nature of graphs triggers substantial irregular memory access since edge connections are distributed in an uneven pattern, bringing extra overhead to training. Consequently, employing the graph cluster structure on GPUs is promising for graph transformer training improvement. There exist some approaches in traditional graph learning [62], [63] to utilize graph cluster, but they are aimed for CPU processing with limited parallelization. [64] also exploits graph cluster but focuses on redundant data loading in GNN computing.

Therefore, to explore the performance benefits of graph cluster on graph attention computing, we incorporate a lightweight node reordering to cluster nodes and improve the spatial locality during attention computation, without changing the connectivity correctness. The key insight is that the proximity of node IDs is more likely to be scheduled to the adjacency of computing units on GPUs where they get processed. In detail, we leverage METIS [65], a community-based graph reordering technique for great cluster locality and ease of integration with parallelism. Specifically, it uses multilevel recursive bipartitioning to divide and coarsen the graph while preserving the essential structure. We optimize the implementation of METIS for a lower cost: we capture the cluster information of graphs and map such locality from the upper level to the underlying GPU kernels, which also enables us to leverage the L1 & L2 cache for refined cluster capturing (later discussed in §III-D).

Specialized Graph Parallelism. To increase the scalability of graph transformers, intuitively TorchGT employs parallelism technologies to dispense the computation across devices. There have been extensive studies in sequence parallelism technologies [37]–[39] for LLMs to support efficient long sequence training. However, current parallelism methods for language models trigger two challenges when applied to graph transformers: (1) failing to leverage graph properties; (2) not supporting various graph encodings. In traditional language models, the input sequence encodes the context of a specific sentence. As such, training the language model requires tokens in the input sequence concatenated in a pre-defined order. In contrast, we observe that for graph learning tasks, *there is no need for graph transformers to predict sequences (graph-level task) or tokens (node-level task) within a position-fixed context*, since they only rely on the graph topology to construct the structural encodings. A motivating example of parallelizing graphs with graph cluster is the graph-level task, where only the global token is critical for inferring the graph type and other node tokens can be arranged in any order.

Based on this insight, we are the ***first*** to design a *Cluster-aware Graph Parallelism* ***specialized for graph transformers***, as shown in Figure 4. Specifically, the raw input sequences $\boldsymbol{S}$ and graph encodings $\boldsymbol{B}$ are randomly partitioned across P devices. Each local sub-sequence $\boldsymbol{S}_{sub}$ and sub-encodings are projected to local matrices: $\boldsymbol{Q}_{sub}, \boldsymbol{K}_{sub}, \boldsymbol{V}_{sub}, \boldsymbol{B}_{sub} \in \mathbb{R}^{\frac{S}{P} \times d}$, assuming they have the same dimensionality. Then in each graph transformer layer, all subspaces are combined together into complete matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$, and $\boldsymbol{B}$ via the highly efficient all-to-all collective communication operation. All-to-all operation owns an advantage over other communication operations (e.g., all-gather and reduce-scatter) in terms of

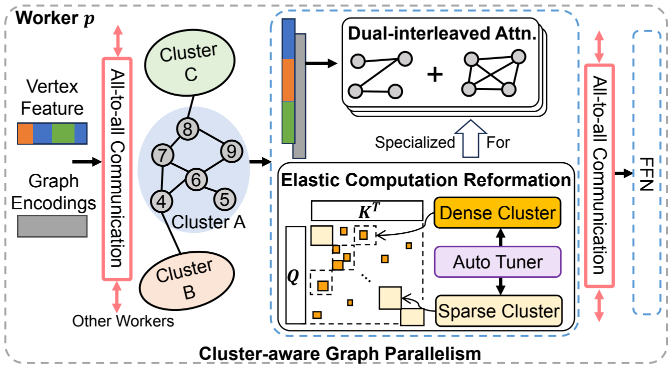

Fig. 4: Detailed training process on one worker with TORCHGT, which includes three key components.

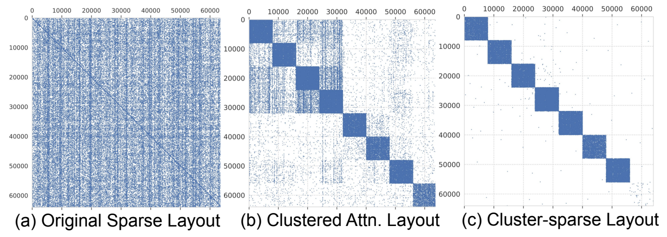

Fig. 5: Three attention layouts after *Dual-interleaved Attention*, *Cluster-aware Graph Parallelism* and *Elastic Computation Reformation* respectively. (c) is obtained by compacting elements to adjacent neighbors inside clusters.

much smaller communication volume and overall better scalability, which is also proved in [38]. All-to-all gathers matrices in sequence dimension and splits in the head, resulting in $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$, and $\boldsymbol{B} \in \mathbb{R}^{S \times \frac{d}{P}}$. Now that since matrices are complete in the sequence dimension, TORCHGT reorganizes the layout according to the clustering nature of graphs discussed before. Then the *Dual-interleaved Attention* conducts attention computation in the clustered layout, exemplified as blue, orange and green rectangles in Figure 4. After attention computation, another all-to-all transforms the output tensor back to subspace $\boldsymbol{S}'_{sub}$ for subsequent operators such as FFN and layer normalization in the graph transformer layer.

Communication Complexity. Thanks to all-to-all, *Cluster-aware Graph Parallelism* has low communication volume and scales exactly well with more servers. Given hidden size d, sequence length S, and parallelism degree P, TORCHGT performs all-to-all with a total message size $3Sd$ before the attention computation, and another all-to-all for attention output with size Sd. Therefore, TORCHGT performs two all-to-alls with communication volume per GPU of $4Sd/P$ and communication complexity of $O(S/P)$, while other operations like all-gather have communication complexity of $O(S)$. Thus, TORCHGT has better scalability with increasing parallelism degree on extremely long sequences.

In summary, compared with sequence parallelism methods for LLMs, our *Cluster-aware Graph Parallelism* favors the graph transformer architecture in several aspects. First, all-to-all gathers in sequence dimension, leading to exactly integrated graph topology, which the topology-induced sparse pattern in §III-B can be perfectly applied to. Second, the graph encodings $\boldsymbol{B}$ share the same sparse layout as attention mapping so the parallelism of graph transformers only brings a trivial memory footprint and communication overhead, thus facilitating model scalability and ensuring memory efficiency.

D. Elastic Computation Reformation

Cluster Sparsity. The topology-induced attention pattern can significantly reduce the computation cost, but also leading to substantial irregular memory access due to the highly skewed nature of graphs. Figure 5(b) gives an example with the cluster dimensionality of 8 and sequence length of 64K. We observe that only the diagonal clusters in the clustered adjacency matrix appear in dense patterns most and show lower sparsity, which can benefit from locality since nodes in each cluster are close to each other. On the other hand, other clusters appear highly sparse patterns and more irregular shapes (denoted as *sparse cluster*). Accessing the embeddings of computation like aggregation in this pattern requires a large number of atomic operations. Consequently, those remaining irregular clusters still engender heavy overhead. To exemplify, training Graphormer on ogbn-arxiv (S=64K) in Figure 5(b) takes 375 ms per epoch, while its dense counterpart only takes 81ms.

To further reduce the memory access latency, we propose a memory-efficient *Elastic Computation Reformation* which introduces the cluster sparsity. Motivated by the block-sparse pattern in [36], [46], [47], we reformat the clustered layout in Figure 5(b) to a fine-grained cluster-level fashion in Figure 5(c). Specifically, as shown in Figure 4, for each sparse cluster, TORCHGT transfers the scattered edges inside it to multiple substructures of compact and adjacent edge connections, which is denoted as *sub-blocks*. The transferred *dense cluster* can have multiple randomly scattered sub-blocks, the number of which is decided by the number of real edges in the cluster and the dimension of sub-block d_b. Note that there will be totally k^2 clusters for the whole layout with the cluster dimensionality of k. Figure 5(c) depicts the cluster-sparse layout with k=8, in which most sparse clusters are transferred to dense ones with tight sub-blocks. This cluster sparsity offsets the downside of irregular memory access incurred by the topology-induced pattern.

Transfer Strategy. Applying static cluster-sparse transferring will result in model quality degradation since the cluster sparsity changes the original graph structure by modifying edges. Some performance-related information (e.g., convergence) for model training is only available at runtime. Without considering the runtime information, the system will suffer from an inferior model accuracy or inefficient memory access.

Thus, TORCHGT designs the following two strategies:

- *Indolent Transferring.* TORCHGT only transfers clusters that are extremely sparse and irregular. Although such an inactive way may miss some optimization opportunities, it can refrain from model quality decline and be more portable. Concretely, TORCHGT only transfers sparse clusters whose sparsity value β_C is smaller than that of the current whole graph β_G. Note that the sparsity value refers to the propor-

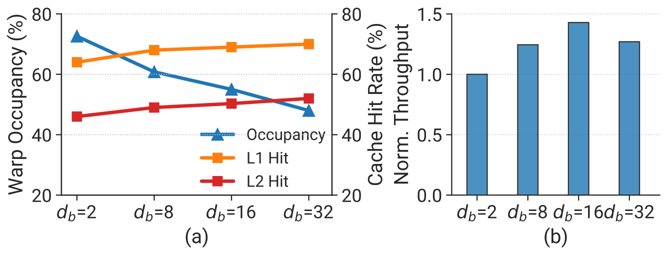

Fig. 6: (a) Profiled hardware statistics of GPU when computing in different sub-block size d_b. The ideal d_b considers both load balance and cache hit rate. (b) Computation throughput of the indexing kernel in different d_b normalized on that of d_b=2.

tion of **nonzero** elements in the whole adjacency matrix.

- *Elastic Transferring.* We dynamically adjust the amount of transferred dense clusters along the training. First, we set a threshold value β_{thre} for controlling cluster-sparse transfer. If the sparsity of a cluster β_C is smaller than the threshold β_{thre}, TORCHGT transfers it to a dense cluster. To decide the value of β_{thre} in each training epoch, we design an *Auto Tuner* in the next part for modeling β_{thre}.

Hyperparameter Modeling. The hyperparameters can be tuned to accommodate various graph patterns. We design an *Auto Tuner* to dynamically select the hyperparameters k, d_b and β_{thre}, and formulate the analytical model.

(1) Cluster dimensionality k, sub-block dimension d_b. We can leverage GPU L1 and L2 caches to improve the memory access locality of sub-block computation. In this way, smaller sub-blocks can enjoy the data locality benefit from the L1 cache while larger clusters can enjoy the locality from the larger L2 cache. Specifically, we determine k as: $k = \lfloor\sqrt{\frac{Q_{L2}}{id}}\rfloor, i \in \mathbb{N}$, where Q_{L2} is the L2 cache size and d is model hidden dimension. To determine the sub-block dimension d_b, we profile the computation throughput and some hardware statistics of the indexing kernel w.r.t. different d_b values. Figure 6(a) shows the workload balance in GPU computing unit downgrades (the lower warp occupancy, the worse balance) as d_b increases, while both L1 & L2 cache hit rates increase. Thus, there exists a trade-off between these two metrics in deciding d_b. Moreover, Figure 6(b) also demonstrates the values of obtaining the optimal computation throughput lie in the middle range. Both cases suggest the middle value is the ideal choice. For example, for RTX 3090 GPU and model hidden dimension of 64, we fit k=8 and d_b=16. These two values can also be selected by users.

(2) Transfer threshold β_{thre}. Motivated by [7], to estimate the convergence, *Auto Tuner* tracks a running average loss $F_t = 0.9F_{t-1} + 0.1\mathcal{L}_t$, where $\mathcal{L}_t$ is the loss at epoch t. Considering the training throughput, a Loss Descent Rate (LDR) is defined as $LDR_t = \frac{F_t - F_{t-1}}{et_t}$, where et_t is the t-th epoch training time. At the beginning $\beta_{thre,0}$ is initialized as β_G from the set $\{0, \beta_G, 1.5\beta_G, 5\beta_G, 7\beta_G, 10\beta_G, 1\}$, which is developed by profiling different datasets. When $LDR_t \geq LDR_{t-\delta}$ for some $\delta \in \mathbb{N}$ (here we use $\delta = 10$) which specifies the range of epochs for LDR comparisons, TORCHGT heuristically determines the current β_{thre} suffices to reduce the loss. Then *Auto Tuner* increases β_{thre} to the next value in the set to gain higher speed. On the other hand, $LDR_t < LDR_{t-\delta}$ in δ epochs denotes the training is about to converge or too many errors are introduced by quantization. In this case, *Auto Tuner* reduces β_{thre} to the previous value from the set to enable more stable and accurate training.

TABLE III: Detailed information of datasets in evaluation.

Node-level				
Datasets	**# Nodes**	**# Edges**	**# Feats**	**Task**
Amazon [24]	1,598,960	132,169,734	200	107-class Classif.
ogbn-arxiv [23]	169,343	1,166,243	128	40-class Classif.
ogbn-products [23]	2,449,029	61,859,140	100	47-class Classif.
ogbn-papers100M [23]	111,059,956	1,615,685,872	128	Binary Classif.
Graph-level				
Datasets	**# Graphs**	**Avg. # Nodes**	**Avg. # Edges**	**Task**
ZINC [66]	12,000	23.2	24.9	Regression
ogbg-molpcba [23]	437,929	26.0	28.1	128-task Classif.
MalNet [53]	10,833	15,378	35,167	5-class Classif.

TABLE IV: Detailed information of graph transformer models.

Models	# Layers	Hidden Dim.	# Head
Graphormer$_{slim}$ (GPH$_{slim}$)	4	64	8
Graphormer$_{large}$ (GPH$_{large}$)	12	768	32
GT	4	128	8

IV. EVALUATION

We implement TORCHGT atop PyTorch 2.1 [67]. We study the performance of TORCHGT on versatile datasets and graph learning tasks in the following aspects: (1) Efficiency (§IV-A), (2) Convergence (§IV-B), (3) Scalability (§IV-C), and (4) micro-benchmarks and ablation studies to examine the impact of each technique and hyperparameter (§ IV-D).

Datasets and Models. We evaluate TORCHGT on versatile real-world graph datasets with multiple scales. The detailed information is shown in Table III, including both node-level and graph-level tasks. The MalNet dataset is constructed from all categories of the full datasets. We use three classical graph transformer models commonly adopted for evaluation, including Graphormer$_{Slim}$ (GPH$_{Slim}$) [14], Graphormer$_{Large}$ (GPH$_{Large}$) [14], and GT [15]. Note that TORCHGT can also be applied to other graph transformer models. As shown in Table IV, we follow the hyperparameter configurations reported in their original papers as closely as possible.

Baselines. All models cannot be directly trained on selected large graphs. Due to the lack of existing graph transformer systems, we meticulously replicate each model with simple graph parallelism following its original implementation as the vanilla version, denoted as GP-RAW. On the basis of this, we have also developed other variants incorporating FlashAttention [31] denoted as GP-FLASH, and topology-induced sparse attention denoted as GP-SPARSE.

Testbed. Our experiments are performed on two GPU servers. ❶3 GPU servers each with 8 NVIDIA RTX 3090 GPUs (24GB). Intra-server connections (CPU-GPU, GPU-GPU) are based on PCIe 4.0x16 lanes and inter-server connections are via 1Gbps Ethernet. ❷2 servers each with 8 A100 GPUs (80GB) with NVLink and 200Gbps InfiniBand.

TABLE V: Detailed comparison of training speed and test accuracy of methods when training on one 3090 GPU server. OOM means the method runs out of memory. TORCHGT always outperforms GP-FLASH in throughput and accuracy on all the models and datasets. GP-RAW with full attention runs out of memory in all cases.

Model	Method	MalNet		ogbn-papers100m		ogbn-products		ogbn-arxiv		Amazon	
		t_{epoch}(s)	Test Acc.(%)	t_{epoch}(s)	Test Acc.(%)	t_{epoch}(s)	Test Acc.(%)	t_{epoch}(s)	Test Acc.(%)	t_{epoch}/s	Test Acc.(%)
GPH$_{Slim}$	GP-Raw	OOM	-	OOM	-	OOM	-	OOM	-	OOM	-
	GP-Flash	2158.37	90.87	1201.13	90.11	27.69	66.39	0.44	48.25	17.31	63.51
	TORCHGT	195.54(**11.0**×)	92.71	19.15(**62.7**×)	96.82	0.54(**50.8**×)	66.75	0.11(**3.9**×)	53.81	1.00(**17.5**×)	73.10
GPH$_{Large}$	GP-Raw			OOM	OOM	OOM	-	OOM	-	OOM	-
	GP-Flash	OOM		2512.88	96.93	56.51	44.48	3.46	22.11	36.83	73.34
	TORCHGT			654.72(**3.8**×)	98.60	16.10(**3.5**×)	63.06	1.16(**3.0**×)	42.38	11.07(**3.3**×)	73.75
GT	GP-Raw	OOM	-	OOM	-	OOM	-	OOM	-	OOM	-
	GP-Flash	1426.24	74.54	1235.02	88.86	28.80	66.20	0.50	53.98	8.88	69.07
	TORCHGT	242.58(**5.9**×)	90.13	26.33(**46.9**×)	89.60	0.79(**36.3**×)	82.11	0.09(**5.3**×)	56.72	0.76(**11.7**×)	72.98

TABLE VI: Training time per epoch of trianing GPH$_{Slim}$ on one A100 server. TORCHGT can still improve training efficiency compared with GP-FLASH.

Model	Method	MalNet	ogbn-papers100m	ogbn-products	Amazon
		t_{epoch}(s)	t_{epoch}(s)	t_{epoch}(s)	t_{epoch}(s)
GPH$_{Slim}$	GP-Flash	668.23	492.79	5.34	3.43
	TORCHGT	160.61(**4.2**×)	244.07(**2.1**×)	2.86(**1.9**×)	1.69(**2.0**×)

A. *End-to-end Training Throughput*

We compare the end-to-end training time per epoch and test accuracy of TORCHGT with all baselines on one server, as shown in Table V. The sequence length is 256K for GPH$_{Slim}$ and GT, and 32K for GPH$_{Large}$. When training on ogbn-arxiv, we set the sequence length to 64K for GPH$_{Slim}$ and GT. The speedup in the bracket is the relative throughput of each method on the basis of GP-FLASH. In each training task, we treat the first 10 epochs as the warmup stage and only record statistics afterward. TORCHGT substantially outperforms GP-FLASH by 3.3∼62.7×. This is mainly because TORCHGT significantly reduces the computation complexity of the attention module. Additionally, GP-RAW runs out of memory (OOM) on all datasets under the current sequence lengths due to its $O(N^2)$ memory complexity of the attention module. For instance, GP-RAW requires over 200GB memory to store the attention score, i.e., $QK^{\top}$, of only one attention head for the ogbn-products dataset. We also conduct evaluations of GPH$_{Large}$ on one A100 server as shown in Table VI. TORCHGT still shows impressive acceleration and outperforms GP-FLASH up to 4.2× on such frontier equipment. In summary, TORCHGT realizes efficient training of graph transformers with a marvelous improvement.

The speedup difference is mainly related to the input graph topology and model structure under long sequences. If the input graph is very sparse (up to 99% sparsity), *Dual-interleaved Attention* first boosts attention by a large margin. If an obvious clustering pattern exists in the graph, then attention can be further accelerated by 2∼3× with the other two modules. Since we mainly improve attention computation, the more proportion it accounts in total model computation, the higher speedup on epoch time. For instance, in Table V GPH$_{Slim}$ achieves notable speedup on papers100M owing to the above. Ogbn-arxiv shows a smaller speedup on all models since it has poorer sparsity and cluster property.

TABLE VII: Training throughput and test accuracy of methods. The accuracy of GP-FLASH decreases because of BF16.

		GP-FLASH	TORCHGT-BF16	TORCHGT-FP32
ogbn-arxiv	t_{epoch}(s)	0.44	**0.08**	0.11
	Test Acc.(%)	48.25	48.29	**53.81**
Amazon	t_{epoch}(s)	17.31	**0.60**	1.00
	Test Acc.(%)	63.51	63.58	**73.10**

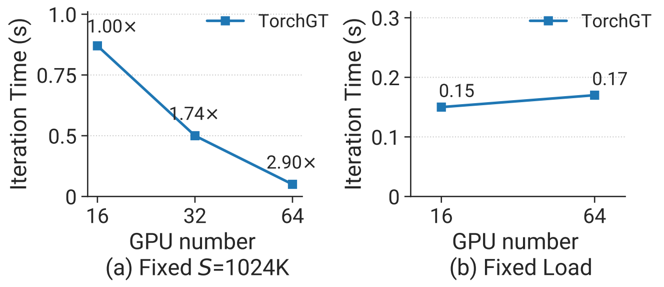

Fig. 7: Scalability results of TORCHGT in training GPH$_{Slim}$ on ogbn-products on many A100 servers. (a) With fixed sequence length, throughput reduces almost linearly. (b) With fixed computational load per GPU, throughput remains well.

B. *Model Convergence*

We examine the model accuracy and convergence curves of TORCHGT on various models in Table V and Figure 8. Table V summarizes the test accuracy achieved by all systems on three models. On large-scale datasets, TORCHGT gives higher model accuracy while GP-RAW runs out of memory. GP-FLASH harms the model accuracy in some datasets, e.g., Malnet and Amazon since FlashAttention only supports FP16/BF16 precision [31] in computing which may downgrade model convergence compared to FP32 precision. In contrast, TORCHGT supports FP32 precision without compromising model accuracy. To better validate this, we compare the training throughput and test accuracy of GP-FLASH and TORCHGT-BF16 in Table VII. On BF16, TORCHGT obtains similar accuracy with FlashAttention, indicating the accuracy drop of FlashAttention is mainly caused by reduced precision. TORCHGT even achieves higher speedup with BF16, but we choose to use FP32 since it gives higher accuracy with notable speedup. From Figure 8, we also see that GP-FLASH converges to low accuracy and lead to far slower convergence than our system, verifying it preserves model quality well.

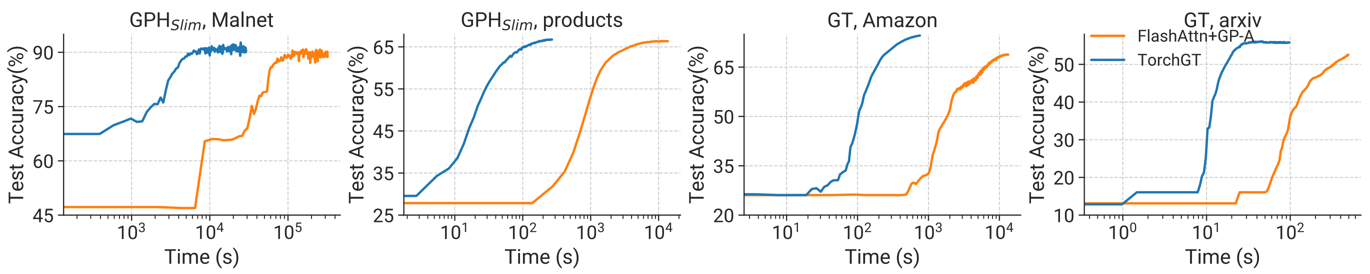

Fig. 8: The convergence curve comparisons of TORCHGT and GP-FLASH on different models and datasets.

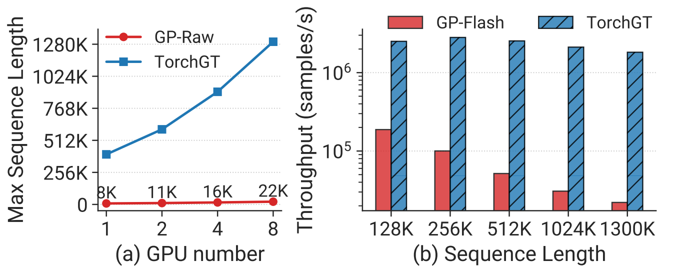

Fig. 9: Scalability experiments of training GPH$_{Slim}$ on ogbgn-products. (a) The supported maximum sequence length w.r.t. GPU number. (b) Training throughput w.r.t. sequence length. In both cases TORCHGT shows greater scalability than others.

C. System Scalability

Training Throughput on Multiple Servers. First, we evaluate the training throughput of TORCHGT on multiple A100 servers to validate it also scales out well with more servers. As shown in Figure 7, we conduct two sets of scalability evaluations on up to 8 servers(each with 8 A100 GPUs) with extremely long sequences and record the sequence training time on the ogbn-products dataset. In Figure 7(a), we fix the sequence length to 1024K and increase the server number. We can see TORCHGT still obtains notable speedup when scaling to more servers. Especially, when the GPU count is doubled, the training throughput correspondingly increases by almost $1.7\times$, indicating a certain degree of scalability. In Figure 7(b), we fix the computational load per GPU when increasing the sequence length from 256K to 512K. Note that when doubling the sequence length, we need $4\times$ GPUs than before to keep the same computational load per GPU(attention calculation is proportional to S^2/P). In this case, TORCHGT achieves approximately the same throughput on each GPU as before, also verifying good scalability.

Sequence Length w.r.t. Number of GPUs. We examine the maximum sequence length of GPH$_{Slim}$ that can be trained on $1\sim8$ GPUs with TORCHGT in Figure 9(a). Note that GP-RAW employs standard full attention. We can see the maximum sequence length of TORCHGT can reach up to 1.3M on 8 GPUs. It also enables the sequence length of 400K with only 1 GPU, substantially $50\times$ larger than that of GP-RAW. Moreover, the sequence length of TORCHGT almost scales linearly w.r.t. the number of GPUs, while the maximum sequence length GP-RAW can support nearly remains unchanged with the growth of GPU numbers. With 8 GPUs, TORCHGT supports 1.3M in

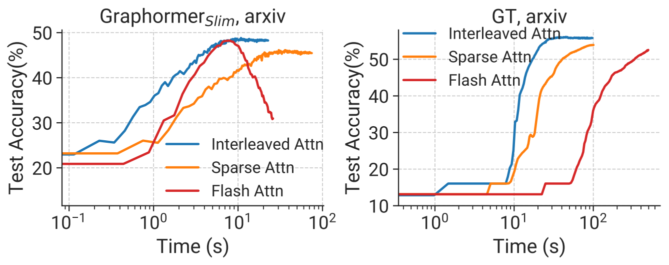

Fig. 10: Convergence comparisons of different attentions: our interleaved attention, FlashAttention and sparse attention.

length while GP-RAW only supports 22K in length.

Throughput w.r.t. Sequence Length. We further compare the training throughput of TORCHGT and GP-FLASH under sequence lengths varying from 128K to 1300K in Figure 9(b). We fix the number of GPUs to 8 and report the throughput as samples per second. Figure 9(b) shows that the training throughput of GP-FLASH sharply decreases from 1.9×10^5 samples/s to 2.2×10^4 samples/s when the sequence length increases. The speed degradation of GP-FLASH mainly comes from the computation bottleneck of FlashAttention with $O(N^2)$ complexity. In contrast, TORCHGT maintains the training throughput at around 2.5×10^6 samples/s by significantly reducing the attention computation costs (in §III-B).

D. Micro-benchmarks

We explore the effects of each component in TORCHGT via ablation studies and perform sensitivity analysis of the introduced hyperparameters on one 3090 GPU server.

Impact of *Dual-interleaved Attention*. After employing the topology-induced attention and interleaving mode introduced in §III-B, we investigate their effect on model quality.

(1) On large-scale graphs. We measure the convergence curves of GPH$_{Slim}$ and GT on ogbn-arxiv, and compare the convergence of interleaved attention with that of FlashAttention and the sparse variant in Figure 10. The model with interleaved attention shows faster convergence than the other two and finally converges to higher accuracy, verifying that the interleaved attention improves computation efficiency while displaying great convergence.

(2) On small graphs. Since the raw graph transformer models fail to be trained on large graphs, we further evaluate the convergence of interleaved attention on small graphs in Figure 11. Sparse attention shows the worst convergence rate while full attention has the best. The model with interleaved

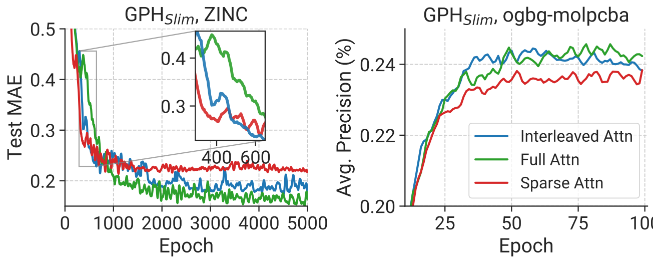

Fig. 11: Convergence curves of different attentions: our interleaved attention, full and sparse attention on small graphs.

attention converges to nearly the same as the model with full attention and obviously outperforms the sparse variant in both convergence speed and final test score.

Impact of *Elastic Computation Reformation*. We particularly examine the impact of the *Elastic Computation Reformation* module in TORCHGT, FlashAttention, and sparse attention w.r.t. the sequence length and the model hidden dimension on one GPU. Note that we implement the sparse variant with the pure topology-induced attention pattern.

(1) Attention computation time w.r.t. sequence length. In Figure 12(a), we first evaluate the speed of the attention module when varying the sequence length from 64K to 512K. We use the model hyperparameter setting in GPH$_{Slim}$ and record the computation time of the attention module in each method. Clearly, as the sequence length increases, the computation time of FlashAttention grows quadratically, resulting in heavy training slowdown. The sparse attention improves the computation speed a bit, but shares a similar computation speed as FlashAttention when the sequence length is small. In contrast, TORCHGT essentially improves the computation efficiency by up to 103.4× compared to FlashAttention. It is even faster than the sparse attention largely, validating its effectiveness in reducing irregular memory access with our cluster-sparse pattern.

(2) Attention computation time w.r.t. hidden dimension. We fix the sequence length to 256K and change the hidden dimension from 64 to 256. The computation time of the attention module is recorded in Figure 12(b). When the model size increases, TORCHGT still largely outperforms FlashAttention and sparse attention in all cases, owing to the specialized cluster sparsity in *Elastic Computation Reformation* module. We can conclude that FlashAttention shows poorer adaptation on long sequences, compared to its higher tolerance on larger model sizes. This indicates the better scalability of TORCHGT for long sequences and large model sizes.

(3) Sensitivity analysis of transfer threshold. The introduced hyperparameter β_{thre} determines the model performance and efficiency of TORCHGT. As shown in Table VIII, we adopt different values of β_{thre} and record the training time per epoch and test accuracy. Note that a larger β_{thre} means more clusters are transferred to dense ones with sub-blocks. Higher accuracy can be obtained when smaller β_{thre} is adopted, but coming with possibly lower training speed (e.g., for GraphSAGE, 0.368s with $\beta_{thre} = \beta_G$ vs. 0.077s with $\beta_{thre} = 10\beta_G$). Seriously chasing the lowest error ($\beta_{thre} = 0$) or just caring

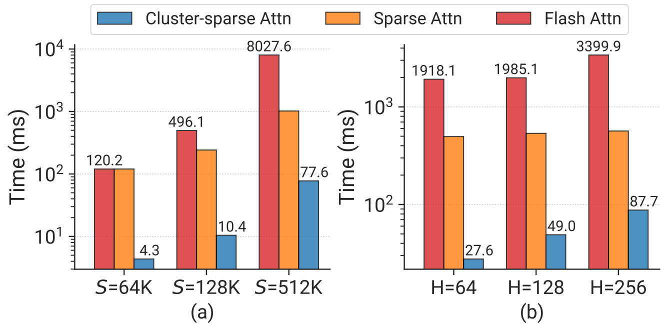

Fig. 12: Computation time of attention modules when training Graphormer on ogbn-products with different (a) sequence lengths and (b) model hidden dimensions when S=256K.

TABLE VIII: Training time per epoch and test accuracy on ogbn-arxiv dataset regarding different transfer threshold β_{thre}.

	β_{thre}	β_G	$1.5\beta_G$	$5\beta_G$	$7\beta_G$	$10\beta_G$	TORCHGT
GPH$_{Slim}$	t_{epoch}**(s)**	0.368	0.257	0.088	0.087	0.077	0.114
	Test Acc.(%)	53.34	54.19	53.82	50.84	48.31	53.81
GT	t_{epoch}**(s)**	0.098	0.090	0.089	0.084	0.071	0.093
	Test Acc.(%)	56.70	56.84	56.51	53.65	45.95	56.72

about the highest training throughput ($\beta_{thre} = 1$) is not the best choice to fully utilize the benefits of the cluster-sparse pattern. Choosing different values always creates a trade-off between efficiency and accuracy and further analysis on the value choice can be done in the future. Currently, we suggest $\beta_{thre} = 5\beta_G$ for better balance.

E. Pre-processing Cost

We record the pre-processing cost versus model convergence time on both tasks to understand how much extra time is brought by TORCHGT. The proportion is 5.2s (5.4%) versus 91.2s (94.6%) for ogbn-arxiv, and 239.7s (2.0%) versus 11732.4s (98.0%) for MalNet. The overhead only occupies less than 5.4% of the total training time on all epochs, which is acceptable compared with the huge model convergence time.

V. RELATED WORK

As a new kind of graph learning algorithms outperforming traditional GNNs, many graph transformer architectures have arisen in recent years. Among them, models [14], [16], [27], [28], [30] utilize standard attention as foundation encoders to capture the all-pair interactions between nodes, leading to quadratic computation complexity. Some other works [16], [32], [57] adopt sampling or pooling methods which only select a subset of nodes to be trained at each iteration, without reducing the computation complexity. [17], [20], [21] use self-defined adapted attention with poor generality and scalability. As for system optimizations for transformers, sparse attention [35], [36], [55], [56] has been widely studied in NLP area for linear complexity. Borrowing this, [15], [25], [26] directly apply graph topology in the attention computation. Besides, multiple works for LLM [37]–[40] split the input sentence sequences and train distributedly for larger scalability.

VI. Conclusion

To conclude, TorchGT reveals challenges and opportunities in training graph transformer on large graphs, with our efforts in designing a scalable and efficient training system.

Acknowledgments

We sincerely thank our anonymous SC reviewers for their valuable comments. This research is supported under the National Key R&D Program of China (2022ZD0160201) and the RIE2020 Industry Alignment Fund - Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s).

References

[1] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, ser. ICLR '16, 2016.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '17, 2017.

[3] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Li, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, ser. ICLR '18, 2018.

[4] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *CoRR*, vol. abs/1810.00826, 2019.

[5] P. Li, Y. Guo, Y. Luo, X. Wang, Z. Wang, and X. Liu, "Graph neural networks based memory inefficiency detection using selective sampling," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. IEEE Press, 2022.

[6] Q. Sun, Y. Liu, H. Yang, R. Zhang, M. Dun, M. Li, X. Liu, W. Xiao, Y. Li, Z. Luan, and D. Qian, "Cognn: efficient scheduling for concurrent gnn training on gpus," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. IEEE Press, 2022.

[7] M. Zhang, Q. Hu, C. Wan, H. Wang, P. Sun, Y. Wen, and T. Zhang, "Sylvie: 3d-adaptive and universal system for large-scale graph neural network training," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 3823–3836.

[8] S. Abadal, A. Jain, R. Guirado, J. Lpez-Alonso, and E. Alarcn, "Computing graph neural networks: A survey from algorithms to accelerators," *CoRR*, vol. abs/2010.00130, 2021.

[9] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI Conference on Artificial Intelligence*, ser. AAAI '20, 2020.

[10] U. Alon and E. Yahav, "On the bottleneck of graph neural networks and its practical implications," in *International Conference on Learning Representations*, ser. ICLR '21, 2021.

[11] J. Topping, F. D. Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," in *International Conference on Learning Representations*, ser. ICLR '22, 2022.

[12] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, ser. AAAI '19, 2019.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, . Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '17, 2017.

[14] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" in *Advances in Neural Information Processing Systems*, ser. NeurIPS '21, 2021-12-06.

[15] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *CoRR*, vol. abs/2012.09699, 2021.

[16] J. Chen, K. Gao, G. Li, and K. He, "Nagphormer: A tokenized graph transformer for node classification in large graphs," in *International Conference on Learning Representations*, ser. ICLR '23, 2023.

[17] Q. Wu, W. Zhao, Z. Li, D. Wipf, and J. Yan, "Nodeformer: A scalable graph structure learning transformer for node classification," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '22, 2022.

[18] D. Kreuzer, D. Beaini, W. L. Hamilton, V. Ltourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '21, 2021.

[19] D. Chen, L. O'Bray, and K. Borgwardt, "Structure-aware transformer for graph representation learning," in *Proceedings of the 39th International Conference on Machine Learning*, ser. ICML '22, 2022, pp. 3469–3489.

[20] Q. Wu, C. Yang, W. Zhao, Y. He, D. Wipf, and J. Yan, "Difformer: Scalable (graph) transformers induced by energy constrained diffusion," in *International Conference on Learning Representations*, ser. ICLR '23, 2023.

[21] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, and J. Yan, "Simplifying and empowering transformers for large-graph representations," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '23, 2023.

[22] Z. Wu, P. Jain, M. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '21, 2021-12-06.

[23] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '20, 2020.

[24] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *Proceedings of the 25th International Conference on World Wide Web*, ser. WWW '16, 2016.

[25] L. Rampek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '22, 2022-12-06.

[26] H. Shirzad, A. Velingker, B. Venkatachalam, D. J. Sutherland, and A. K. Sinop, "Exphormer: Sparse transformers for graphs," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML '23, 2023, pp. 31 613–31 632.

[27] L. Chanussot, A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu, A. Palizhati, A. Sriram, B. Wood, J. Yoon, D. Parikh, C. L. Zitnick, and Z. Ulissi, "The open catalyst 2020 (oc20) dataset and community challenges," *CoRR*, vol. abs/2010.09990, 2021.

[28] Z. Fan, T. Chen, P. Wang, and Z. Wang, "Cadtransformer: Panoptic symbol spotting transformer for cad drawings," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 986–10 996.

[29] E. Min, Y. Rong, T. Xu, Y. Bian, P. Zhao, J. Huang, D. Luo, K. Lin, and S. Ananiadou, "Neighbour interaction based click-through rate prediction via graph-masked transformer," *CoRR*, vol. abs/2201.13311, 2022.

[30] Y.-L. Liao and T. Smidt, "Equiformer: Equivariant graph attention transformer for 3d atomistic graphs," in *International Conference on Learning Representations*, ser. ICLR '23, 2023.

[31] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. R, "Flashattention: Fast and memory-efficient exact attention with io-awareness," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '22, 2022-12-06.

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

[33] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, ser. ICLR '18, 2018.

[34] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *International Conference on Learning Representations*, ser. ICLR '20, 2020.

[35] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser, D. B. Belanger, L. J. Colwell, and A. Weller, "Rethinking attention with performers," in *International Conference on Learning Representations*, ser. ICLR '23, 2023.

[36] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird:

Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[37] S. Li, F. Xue, C. Baranwal, Y. Li, and Y. You, "Sequence parallelism: Long sequence training from system perspective," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 2391–2404. [Online]. Available: https://aclanthology.org/2023.acl-long.134

[38] S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, S. L. Song, S. Rajbhandari, and Y. He, "Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models," *CoRR*, vol. abs/2309.14509, 2023.

[39] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," in *Proceedings of Machine Learning and Systems*, ser. MLSys '23, 2023.

[40] H. Liu, M. Zaharia, and P. Abbeel, "Ring attention with blockwise transformers for near-infinite context," *CoRR*, vol. abs/2310.01889, 2023.

[41] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, and P. Kalnis, "Mizan: a system for dynamic load balancing in large-scale graph processing," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. Association for Computing Machinery, 2013.

[42] H. Liu and H. H. Huang, "Enterprise: breadth-first graph traversal on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. Association for Computing Machinery, 2015.

[43] A. Sala, H. Zheng, B. Y. Zhao, S. Gaito, and G. P. Rossi, "Brief announcement: revisiting the power-law degree distribution for social graph analysis," in *Proceedings of the 29th ACM SIGACT SIGOPS symposium on Principles of distributed computing*, ser. PODC '10, 2010.

[44] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '20, 2020.

[45] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozire, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," *CoRR*, vol. abs/2302.13971, 2023.

[46] S. Narang, E. Undersander, and G. Diamos, "Block-sparse recurrent neural networks," *CoRR*, vol. abs/1711.02782, 2017.

[47] J. Qiu, H. Ma, O. Levy, W.-t. Yih, S. Wang, and J. Tang, "Blockwise self-attention for long document understanding," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020.

[48] Z. Zhang, X. Wang, C. Guan, Z. Zhang, H. Li, and W. Zhu, "Autogt: Automated graph transformer architecture search," in *International Conference on Learning Representations*, ser. ICLR '23, 2023.

[49] M. S. Hussain, M. J. Zaki, and D. Subramanian, "Global self-attention as a replacement for graph convolution," *CoRR*, vol. abs/2108.03348, 2022.

[50] J. Zhang, H. Zhang, C. Xia, and L. Sun, "Graph-bert: Only attention is needed for learning graph representations," *CoRR*, vol. abs/2001.05140, 2020.

[51] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Z.-H. Zhou, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2021, pp. 1548–1554, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2021/214

[52] L. M. S. Khoo, H. L. Chieu, Z. Qian, and J. Jiang, "Interpretable rumor detection in microblogs by attending to user interactions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, ser. AAAI '20, 2020.

[53] S. Freitas, Y. Dong, J. Neil, and D. H. Chau, "A large-scale database for graph representation learning," *arXiv preprint arXiv:2011.07682*, 2020.

[54] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. Los Alamitos, CA, USA: IEEE Computer Society, mar 2021, pp. 97–110. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HPCA51647.2021.00018

[55] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *CoRR*, vol. abs/2006.04768, 2020.

[56] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *International Conference on Learning Representations*, ser. ICLR '20, 2020.

[57] C. Liu, Y. Zhan, X. Ma, L. Ding, D. Tao, J. Wu, and W. Hu, "Gapformer: Graph transformer with graph pooling for node classification," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, E. Elkind, Ed. International Joint Conferences on Artificial Intelligence Organization, 8 2023, pp. 2196–2205, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2023/244

[58] C. Yun, Y.-W. Chang, S. Bhojanapalli, A. S. Rawat, S. Reddi, and S. Kumar, "O(n) connections are expressive enough: Universal approximability of sparse transformers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 13 783–13 794, 2020.

[59] "Hamiltonian path," https://mathworld.wolfram.com/HamiltonianPath.html, 2020.

[60] C. Lee and B. Sudakov, "Dirac's theorem for random graphs," *CoRR*, vol. abs/1108.2502v3, 2012. [Online]. Available: https://arxiv.org/abs/1108.2502

[61] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[62] M. E. J. Newman, "Spectral methods for community detection and graph partitioning," *Phys. Rev. E*, vol. 88, p. 042822, Oct 2013. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.88.042822

[63] "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, no. 12, pp. 1519–1534, 2000, graph Partitioning and Parallel Computing.

[64] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, "GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on GPUs," in *15th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI '21. USENIX Association, 2021, pp. 515–531.

[65] K. George and K. Vipin, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.

[66] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *Journal of Machine Learning Research*, vol. 24, no. 43, pp. 1–48, 2023.

[67] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '19, 2019.