



UniTG: A Unified System for Efficient and Seamless Textual Graph Learning

Meng Zhang^{1,2} Zhisheng Ye³ Qiyu Liu^{4,*} Jingshu Peng⁵ Tianwei Zhang¹

¹Nanyang Technological University ²S-Lab, NTU ³Peking University

⁴Southwest University ⁵Hong Kong University of Science and Technology

{meng.zhang,tianwei.zhang}@ntu.edu.sg, yezhisheng@pku.edu.cn, qyliu.cs@gmail.com, jpengab@cse.ust.hk

ABSTRACT

It has become critical to utilize language models (LMs) for representation learning on text-attributed graphs. They enhance the original graph neural networks (GNNs) by delicately modeling text attributes alongside graph structure learning. Despite these algorithmic breakthroughs, existing LM-based graph learning still fails in practical deployment due to several critical defects, namely time and resource inefficiency, inflexible decoupled architectures, limited model scale, and the omission of graph properties.

In this paper, we propose UniTG, the first unified system that fuses the LM and GNN phases into a single end-to-end procedure through three *co-designed* components spanning the runtime, algorithm, and execution levels. At the *runtime* level, UniTG introduces Affinity-aware Flow Parallelism, exploiting graph affinity to scale the training of large graph neural networks. At the *algorithm* level, a novel Collaborative Learning strategy integrates both text and graph modalities to enable accurate joint training. At the *execution* level, the Streamlined Pipeline Schedule squeezes pipeline bubbles by interleaving LM fine-tuning into the GNN pipeline, boosting overall efficiency and resource utilization. Extensive experiments demonstrate that, compared with state-of-the-art LM-based graph learning systems, UniTG dramatically reduces learning makespan by up to **17.3×** without compromising model quality.

PVLDB Reference Format:

Meng Zhang, Zhisheng Ye, Qiyu Liu, Jingshu Peng, Tianwei Zhang. UniTG: A Unified System for Efficient and Seamless Textual Graph Learning. PVLDB, 19(11): 3273 - 3287, 2026.
doi:10.14778/3836663.3836688

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/zxmeng98/TiGraph>.

1 INTRODUCTION

Graph-structured data has long been prevalent in many real-world scenarios. Among them, a lot of graphs possess textual information, and are often referred to as text-attributed graphs (TAGs) [71]. In TAGs, each node is associated with text descriptions such as documents or sentences, and edges represent relationships between

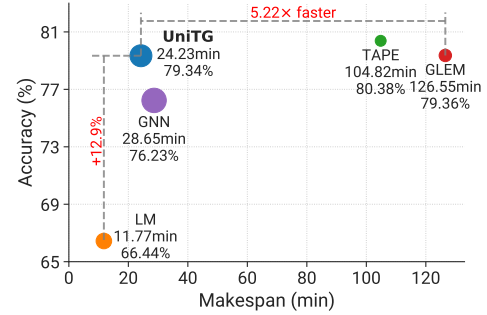


Figure 1: The makespan and learning accuracy comparisons on ogbn-products [31]. Datapoint size is proportional to the average GPU utilization rate. LM backbone is DeBERTa [26] and GNN backbone is RevGCN [39].

these descriptions. For instance, in the ogbn-arxiv dataset [31], a citation network is structured as a TAG, with each node representing a paper and its title and abstract serving as its text attributes. Compared with shallow numerical node attributes, graph topology embedded with textual attributes provides more profound information, significantly enhancing graph representation learning in many tasks such as text classification [11, 75], node/link classification [28, 84], and recommender systems [88].

Graph Neural Networks (GNNs) exhibit impressive performance on many graph learning tasks [12, 22, 42, 63, 66, 79] and have become a standard methodology in this field [12, 22]. However, in most works, GNNs learn graphs only with numerical attributes as inputs, or handle text attributes by transforming them into shallow or hand-crafted features, such as skip-gram [50] or bag-of-words [25] features. These shallow embeddings are limited in the text semantics they can capture, and fail to incorporate graph properties. To overcome this deficiency, recently, language models (LMs) are raised as a promising method for encoding textual attributes. A new branch of LM-based approaches called *LM4Graph* are emerging, which consists of two phases. The first phase leverages pre-trained LMs, such as BERT [13], to generate deeper embeddings that specifically capture the semantic richness of text attributes. In the second phase, GNNs learn structure relationships between nodes on the generated attributes. Due to the great modeling capability on both texts and graph structures, LM4Graph has garnered surging interest in recent years and a large number of approaches have been proposed [11, 15, 28, 32, 84, 88].

As shown in Figure 1, LM4Graph methods like TAPE [28] and GLEM [84] outperform traditional message-passing GNNs (e.g., GCN [36]) on many TAG-related tasks. However, despite the promising accuracy achieved by LM4Graph methods, existing systems fundamentally suffer from a fragmented design that *decouples* language

*Correspondence to Dr. Qiyu Liu.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836688

modeling and graph learning, overlooks graph-specific characteristics, and lacks system-level co-optimization. Current approaches exhibit four key issues:

- I1: Laborious and inflexible designs.** Current LM4Graph designs heavily rely on human labor and expertise knowledge to determine the proper organizations between LM and GNN phases [71, 72, 84, 88] and suitable feature encoding strategies [15, 28, 57, 65], resulting in high integration cost on new TAG tasks. Moreover, many works [84] require frequent checkpoint-reloading and program cold-start for switching phases. These burdensome hand-crafted features are neither flexible for practical deployment nor portable for model generality.
- I2: Agnostic to graph properties.** To accelerate learning, some solutions [11, 28, 76, 84] simply graft existing parallelism strategies, such as data parallelism to the LM phase, without considering graph properties and overall scalability. This graph-agnostic way remains the computation burden, missing optimization opportunities. Others like [15] reduce the computation cost of text encoding via efficient fine-tuning in natural language processing (NLP) era, e.g., LORA [30]. However, simply adopting them without considering the inherent graph structure can result in subpar model performance.
- I3: Decoupled training paradigm.** Current LM4Graph paradigms [11, 15, 28, 32, 57, 84, 88] separately perform LM-based embedding generation and GNN training, with only one phase active at a time. For example, in each step, GLEM [84] first fine-tunes the LM to obtain the encoded features with the GNN phase fixed, and then feeds them into the GNN model. As such, as shown in Figure 1, one needs to go through two distinct training processes sequentially, leading to substantial cumulative learning time (104min and 126min), compared with GNN-only (28min) and LM-only (11min). Moreover, in this one-at-a-time paradigm, the imbalanced workloads across two phases lead to suboptimal hardware utilization. For instance, the LM phase may require 8 GPUs, most of which will be idle during the GNN phase which only needs 2 GPUs, resulting in low GPU utilization in clusters (green and red datapoints).
- I4: Lack of system optimizations.** GNNs rely on massive sparse operations (e.g., sparse matrix-matrix multiplication, SpMM) with highly irregular memory accesses, challenging the optimization of the system throughput. Moreover, with large GNN models, the memory consumption grows rapidly, necessitating a scalable system design and memory optimization. Although there have been plentiful system breakthroughs for deep neural networks in scalable training [23, 33, 34, 51, 85], those optimizations cannot be directly grafted on LM4Graph approaches due to the inconsistent graph learning pipelines.

The aforementioned issues result in performance downsides. Due to versatile self-defined learning procedures, current approaches [11, 15, 28, 32, 57, 84] meet the scalability difficulty when learning efficiency is required. For instance, some works [11, 15, 28] adopt data parallelism for LM phase while keeping GNN phase original on a single GPU, making it arduous to *scale both* at the same time. They also design specific text encoding mechanisms exclusive to themselves. This greatly limits the GNN models and TAG tasks one can apply to very small scale [15, 84, 88], conflicting with the versatile graph representation application scenarios.

To bridge these gaps, we design UniTG, the *first* efficient system for textual graph learning that unifies the two phases into a single end-to-end process, eliminating the need for unnecessary phase switching. Our system abides by four design goals: *efficient*, *seamless*, *accurate* and *portable*. Existing LM4Graph works neither facilitate efficiency by well-designed parallelism from the system perspective nor support flexible extensions to general models and datasets, thus making it challenging to meet those goals. Specifically, the core design of UniTG derives from the following three key insights. *First, the learning of LM4Graph architectures largely benefits from time-sharing computation.* The conventional two-phase execution leads to massive resource idleness and computation starvation. Yet, this very two-phase nature also presents an opportunity for time-sharing, where spare resources are leveraged to improve hardware utilization and overall learning efficiency. *Second, the fine-tuning of LMs on graph data benefits from graph properties.* The LM learns the text embeddings of all graph nodes, while each node actually participates differently in GNN computation. Some nodes are more isolated with few connections with others, thus contributing less to GNN message passing [64, 79]. Thus, embeddings of isolated nodes can be less “deep” for higher tuning efficiency. *Third, the intra-layer GNN computation can be further boosted by data affinity.* Sparse aggregation in GNNs is a key performance bottleneck. Fortunately, graph workloads exhibit significant locality that can be exploited to reduce memory access costs.

In detail, UniTG jointly optimizes computational efficiency and model accuracy for TAG learning from both algorithmic and systems perspectives. Three key components are designed. (1) **Affinity-aware Flow Parallelism** partitions GNN message flows across workers and updates each subset model according to the affinity nature of graphs, thus enhancing system scalability and reducing the irregular memory access latency. (2) **Dual-modality Collaborative Learning** fuses text and graph modalities by synchronizing LM-encoded text attributes into the GNN training pipeline, maximizing model accuracy while accelerating LM fine-tuning via node centrality. To increase GPU utilization, (3) **Streamlined Pipeline Schedule** interleaves LM fine-tuning into idle intervals on workers induced by the pipelined GNN phase (a.k.a., pipeline bubbles).

In summary, we make the following contributions:

- C1:** UniTG is the *first* unified system for LM4Graph learning in one step. Inspired by the unique features of LM4Graph workloads, UniTG exploits the bubble resources to improve hardware utilization and holistic system efficiency.
- C2:** We characterize the key challenges in LM4Graph training and identify graph-specific opportunities for optimizing both LM and GNN phases, including guiding fine-tuning with node centrality and computing nodes based on graph affinity.
- C3:** Through extensive experiments, we demonstrate that UniTG achieves both time and resource efficiency for LM-based graph representation learning. It outperforms SOTA LM4Graph learning systems by up to **17.3×** in training time and **17.8×** in energy consumption while maintaining model accuracy.

We highlight that the key contribution of UniTG is a *holistic system-algorithm co-design*. Each component is specifically adapted to the unique graph properties of LM4Graph and produces synergistic benefits when combined (§4), rather than a straightforward application of existing techniques in isolation.

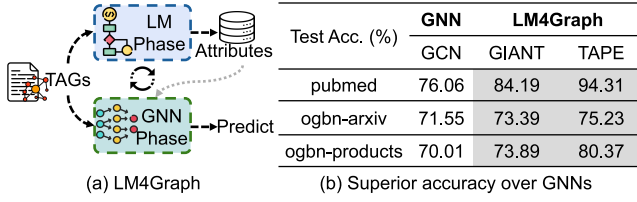


Figure 2: (a) The architecture of LM4Graph methods. (b) LM4Graph methods outperform classical GNNs on TAG-related tasks by a large margin.

2 BACKGROUND

2.1 LM4Graph Learning

LM4Graph (LM-based graph learning) has attracted surging attention in graph representation learning for various tasks on TAGs in recent years [82]. As illustrated in Figure 2(a), current LM4Graph methods typically decompose training into two phases: an LM phase that encodes node text attributes using pre-trained language models, and a GNN phase that performs message passing over the resulting embeddings. Existing approaches follow two main paradigms. Cascaded methods [11, 15, 28, 32, 57, 65, 74, 81, 88] materialize LM-encoded embeddings as checkpoints before launching a separate GNN training task. Another group of works [72, 77, 84] alternate between the LM and the GNN phase, using outputs from one to bootstrap the next. Some other works [14, 73, 76, 80] only adopt LMs to learn node representations and make predictions, which is beyond the scope of this work. As discussed, existing methods have two phases decoupled in a one-at-a-time update pattern, and cold-start the program to switch the phase, which suffers from serious inefficiency and latent errors.

Formally, a TAG can be denoted as $G = (V, A, s_V)$ with nodes $V = \{v_1, \dots, v_N\}$, adjacency matrix $A \in \mathbb{R}^{N \times N}$, and a sequential text attribute $s_v \in D^{L_v}$ for each node $v \in V$, where D is the words dictionary and L_v is the text length.

LM Phase. A pre-trained LM $\mathcal{LM}$ (e.g., BERT [13] or DeBERTa [26]) encodes each node’s text attribute s_v into a d -dimensional embedding vector:

$$h_v = \mathcal{LM}(s_v) \in \mathbb{R}^d. \quad (1)$$

The resulting node embeddings are fed into the GNN as input features for downstream tasks like node classification.

GNN Phase. A GNN updates via the message passing mechanism [21], which first aggregates the feature vectors from a node’s neighbors and then utilizes the aggregated information to update the node’s representation. The l -th layer of a GNN can be recursively expressed as:

$$h_v^l = \phi^l \left(\rho^l \left(\left\{ h_u^{l-1} \mid u \in \mathcal{N}(v) \right\} \right), h_v^{l-1} \right) \in \mathbb{R}^d \quad (2)$$

where h_v^l is the representation of node v at layer l and $\mathcal{N}(v)$ denotes the set of v ’s neighbors. ρ^l is the aggregation function (e.g., sum, mean, etc.) that takes the embeddings of node v ’s neighboring nodes to get an intermediate aggregated result. ϕ^l is the update function that takes the aggregated results and the previous-layer representation to obtain the learned representation. The final representation goes through a fully connected layer and a softmax function for the label prediction.

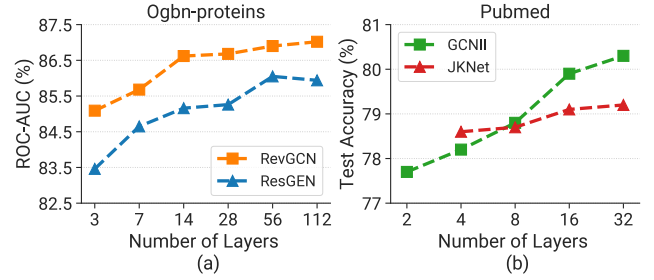


Figure 3: The test score of GNNs when trained with different numbers of layers. Deep layers show better performance.

LM Backbone	GNN Backbone		
	RevGAT	GraphSAGE	GCN
deberta-base (100M)	75.38	74.78	72.07
all-roberta-large-v1 (355M)	75.40	74.87	72.82
e5-large (560M)	75.36	74.79	72.68

Table 1: Test accuracy (%) on ogbn-arxiv of using different LM backbones with UniTG.

2.2 Graph and Language Models for LM4Graph

Necessity of Large GNNs. Conventional GNN models [22, 36] achieve their best performance with 2 or 3 layers, but such shallow architectures limit their ability to extract structural information from high-order neighbors. Also, real-world graphs can be intricate and easily involve millions of nodes [27, 31], on which shallow GNNs show subpar performance [6, 62]. Recent studies [8, 10, 41, 58, 70, 87] demonstrate that with sophisticated architecture designs, deep GNN models achieve SOTA performance over shallow ones on multiple applications [35, 39, 40, 79] such as point cloud segmentation [3, 69] and node classification [7, 41].

For better illustration, Figure 3 shows the impact of model layers on the test score of several representative GNNs. All architectures benefit from deeper layers, and the power of large depth even becomes more evident on a larger graph dataset ogbn-proteins: ResGEN with 112 layers improves the test score by up to 2.5% compared to the shallow models. On the Pubmed dataset, the 32-layer GCNII also outperforms the model with 2 layers by 2.6% of accuracy. These results prove the necessity for deep GNNs of LM4Graphs. However, existing LM4Graph methods [11, 28, 32, 76, 84] face inherent constraints in scaling to deep architectures due to substantial memory demands and prohibitive training overheads. As shown later, UniTG addresses this gap through algorithm-system co-design to enable deep model training.

Sufficiency of Small LMs. Unlike long-context documents tackled by LLMs, node text attributes in graphs are typically concise (50-100 tokens) in real datasets [31]. A fine-tuned small LM thus provides sufficiently discriminative features for graph learning [11, 15, 28, 84], with comparable performance across different LM4Graph architectures [15, 28]. Table 1 confirms this within UniTG: varying the LM size while fixing the GNN backbone yields negligible accuracy differences. Therefore, UniTG is designed to align with existing methods by adopting small LMs and we illustrate our discussion with DeBERTa [26] hereafter. We also discuss the system challenges of integrating larger LMs in §7.

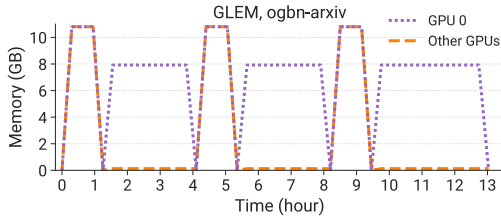


Figure 4: The profiled time and memory consumption of a LM4Graph method on different GPUs.

3 MOTIVATION AND OPPORTUNITIES

3.1 Time-sharing of Two Phases

As discussed earlier, existing LM4Graph works [11, 15, 28, 32, 57, 65, 74, 81, 88] adopt either cascaded or alternating workflows to coordinate LM and GNN phases. However, all decouple the two phases in a one-at-a-time update pattern and cold-start the program for phase switch, which is inefficient and error-prone. Figure 4 demonstrates this on GLEM [84]. Only GPU 0 (purple line) continuously holds both LM and GNN parameters in memory (>8 GB), while all other GPUs remain idle during the GNN phase and activate only when the LM phase recurs. This pattern repeats across training rounds, leaving most GPUs persistently underutilized.

Nonetheless, the unique features of LM4Graph also bring optimization opportunities. (1) *Minor LM portion*. In LM4Graph training, the LM phase usually occupies smaller proportion of the total learning time, making it more tolerant to partial throughput slowdown. We can therefore overlap LM computation with pipeline bubbles, reducing makespan via time-sharing without degrading GNN throughput. (2) *Decreasing resource demand*. We design a Lazy Tuner as described in §4.3, which allows the LM to exit early at the middle layers for certain inputs. The LM thus initially demands more resources but gradually reduces its computational footprint, alleviating kernel contention and allowing us to exploit bubble resources for running more LM encodings.

Model	Epoch Time(s)		
	RevGCN	RevGAT	ResGNN
Original Training	2.59	0.89	0.26
Affinity-aware Training	2.39	0.83	0.23

Table 2: Epoch time of training GNN in affinity-aware way and the original counterpart on ogbn-products.

3.2 Graph Property Awareness

Although LMs have been novelly applied to conduct text encoding for TAGs, existing methods [11, 15, 28, 65, 84] treat them as standard NLP workloads where LMs process all nodes uniformly, applying identical encoding process regardless of their structural importance. However, in practice, nodes contribute unequally to GNN aggregation based on their topological roles [64, 65, 79]. Moreover, existing GNN phases also fail to exploit graph topology in sparse operations such as SpMM.

Graphs exhibit two key properties relevant to LM4Graph’s setting: (1) *data affinity*, and (2) *node centrality*. Table 2 demonstrates that affinity-aware training yields up to 1.13 $\times$ speedup, suggesting that clustering nodes in a graph by affinity improves data locality

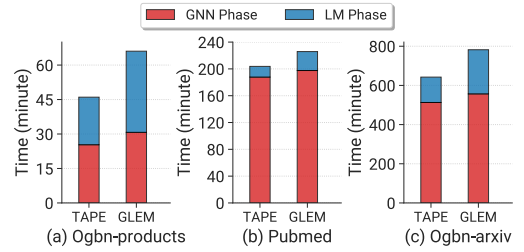


Figure 5: Time breakdown of each phase when training different methods on 3 datasets on GPU-A (details in §5).

and sparse operator (e.g., SpMM) performance [65, 68, 83]. Building on this insight, UniTG implements affinity-aware computation in the GNN phase to accelerate aggregation (§4.2), and centrality-guided LM fine-tuning with adaptive early exit (§4.3). Since both phases operate on the same graph and share the same learning objective, these optimizations apply jointly.

3.3 Interactive Training Fusion

Existing LM4Graph methods adopt decoupled training, incurring substantial overhead because of tedious and repetitive processes. To understand the time bottleneck of current methods, we profile the training time of TAPE [28] and GLEM [84] using DeBERTa-base [26] and RevGAT [39] as the LM and GNN backbones, respectively, across three datasets. Figure 5 shows the detailed training time breakdown. On larger datasets like ogbn-arxiv [31], training extends to 13 hours, with the GNN phase dominating ($>70\%$). This overhead originates from decoupled execution, frequent checkpoint reloads, and cold-start phase transitions, making existing LM4Graph methods much slower than GNN-only and LM-only counterparts. However, our evaluation (Figure 14) shows that feeding LM-encoded embeddings to the GNN concurrently with LM fine-tuning does not compromise convergence or accuracy. This motivates continuous, interleaved training without cold starts, achieving efficiency comparable to GNN-only or LM-only training while matching the accuracy of decoupled LM4Graph methods.

4 SYSTEM DESIGN

4.1 UniTG Overview

Motivated by all observations in §3, we propose UniTG, an algorithm-system co-optimized system tailored for portable LM-based graph learning coherently in one step.

Principles & Goals. UniTG adheres to three design principles:

- (a) *Efficient and accuracy-maintained*. The system jointly optimizes training time and resource utilization across both LM and GNN phases, enabling *deeper* GNN architectures while matching the convergence and accuracy of decoupled baselines (solving **I1**, **I3**, **I4**).
- (b) *Seamless and portable*. UniTG automates the entire LM4Graph learning workflow, eliminating manual phase orchestration and checkpoint management. The LM and GNN phases update concurrently with minimal user intervention (solving **I1**, **I3**).
- (c) *Modular and specialized*. Each component in UniTG works independently and thus can be applied to existing algorithms non-invasively. Tailored to LM4Graph workloads, UniTG leverages graph-specific properties (e.g., affinity, centrality) to scale to large GNN models and generalize across TAG tasks (solving **I2**).

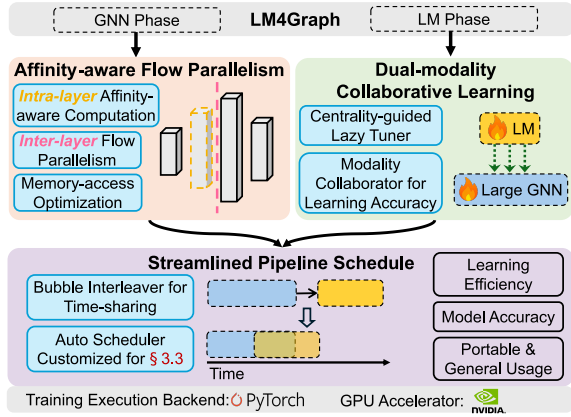


Figure 6: Overview of UniTG architecture and workflow.

System Architecture. Figure 6 shows the architecture and workflow of UniTG, which optimizes LM4Graph training across three layers: runtime, algorithm, and execution.

- **Runtime: Affinity-aware Flow Parallelism.** From the distributed runtime perspective, we design a Flow Parallelism (FP) tailored for GNNs to split the model across workers at the inter-layer level. Leveraging graph affinity, we boost the intra-layer computation by alleviating irregular memory access.
- **Algorithm: Dual-modality Collaborative Learning.** We integrate the training of LM and GNN concurrently as a whole with *Modality Collaborator*. It automatically synchronizes LM-encoded embeddings into the GNN training pipeline, enabling concurrent text and graph modality fusion without compromising accuracy. In addition, a *Lazy Tuner* exploits node centrality to safely accelerate LM fine-tuning via adaptive early exit.
- **Execution: Streamlined Pipeline Schedule.** A *Bubble Interleaver* inserts LM tuning into pipeline bubbles of the GNN phase, thus enabling time-shared execution. We also build an *Auto Scheduler* customized for collaborative learning to adjust the pipeline settings and support extensions to different LM4Graph approaches.

Positioning of UniTG. We emphasize that the novelty of our UniTG lies not merely in individual techniques, but in their graph-specific adaptations and cross-level synergies within the LM4Graph setting. Each component addresses a barrier that precludes direct adoption of existing solutions. (1) Standard pipeline parallelism [33, 52] typically partitions mini-batches into micro-batches; yet in GNNs, excessive partitioning breaks graph connectivity and degrades accuracy (see Figure 8), necessitating affinity-aware partitioning. (2) NLP early exiting [9, 16, 47, 56] relies on sample-level confidence, whereas our Lazy Tuner exploits graph topological property to determine exit depth, which is a simple yet effective indicator for LM4Graph. (3) The Bubble Interleaver also exploits structural properties unique to LM4Graph learning: the coexistence of two heterogeneous phases with complementary resource profiles, where time-sharing is infeasible in single-model training. Moreover, these components are mutually reinforcing. For example, affinity-aware partitioning enhances SpMM locality while preserving microbatch connectivity; and the Lazy Tuner’s progressively shrinking computation frees increasing bubble capacity over time. Detailed ablation studies (§5) isolate the impact and necessity of each design choice.

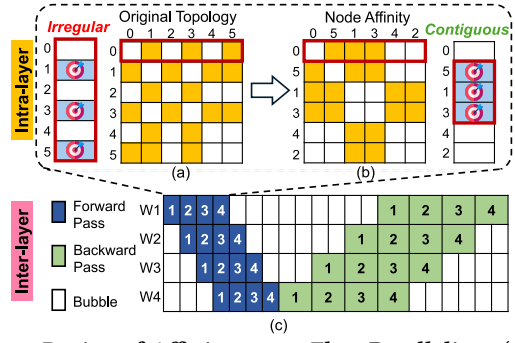


Figure 7: Design of Affinity-aware Flow Parallelism. (a) Original graph topology, (b) affinity-aware topology and (c) Flow Parallelism for a GNN model with 4 pipeline stages.

4.2 Affinity-aware Flow Parallelism

Graph Affinity. Graph affinity [18, 54, 83] characterizes subsets of nodes that are densely intra-connected yet sparsely linked to the remainder of the graph. This graph property directly impacts GNN performance because the aggregation operator in Equation 2 reduces to SpMM, which is a memory-bound kernel whose throughput suffers from irregular accesses. In a sparse matrix, the non-zero elements (NNZs) are stored in a row-major pattern but scattered in a wide range of column indices [5], causing non-contiguous accesses into the dense feature matrix. Thus, affinity-aware partitioning concentrates NNZs within local blocks, improving data locality and GPU cache efficiency and consequently enhancing the performance of SpMM. There exist some approaches [1, 54] in traditional graph learning that also utilize the affinity nature, but they majorly aim for CPU processing with limited parallelization.

Intra-layer Affinity-aware Computation. In light of the above, we exploit the graph affinity to optimize intra-layer GNN computation as shown in the upper part of Figure 7. Specifically, we incorporate a lightweight reordering mechanism to split and rearrange the sparse adjacency matrix into dense blocks, improving spatial locality without changing the original graph connectivity. Figure 7(a) is the adjacency matrix of the original graph, which shows an irregular pattern (yellow blocks). For example, the update of node-0 depends on node-1, -3, and -5 stored in discontinuous memory. In contrast, with affinity-aware node reordering, we obtain the reordered graph in Figure 7(b) and get better data density with more locality in the adjacency matrix. After affinity-aware reordering (Figure 7(b)), the adjacency matrix exhibits denser clusters, and the proximity of node IDs is more likely to be mapped to the adjacency of computing units on GPUs where they get processed. This enables coalesced feature accesses for neighboring nodes within each GNN layer, improving the computing efficiency.

As for the reordering method, we improve upon METIS [20], which partitions the graph and reorders nodes to minimize neighbor access costs, with time and space complexity $O(|V| + |E|)$. We optimize the implementation of METIS for a lower cost: we extract cluster structure from the partitioned graph and propagate this locality information to GPU kernels, enabling efficient cache utilization (L1/L2) for refined cluster capturing.

Inter-layer Flow Parallelism. To scale LM4Graph methods to large GNN models, we split the message flows between GNN layers and dispense computation across devices. Existing approaches

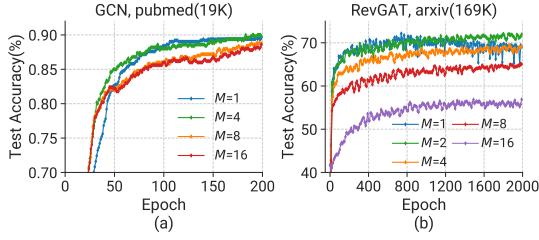


Figure 8: The test accuracy of GNNs in Flow Parallelism when trained with different numbers of microbatches M .

typically distribute only the LM phase, treating the GNN phase as a primitive status. This asymmetry yields fragmented pipelines and limits end-to-end optimization. Other works on parallelism techniques [23, 34, 37, 44, 51] (e.g., GPipe [33]) support large-scale model training. However, directly applying these methods to TAG learning presents two fundamental challenges: (1) *not supporting graph inputs*; (2) *deteriorating model accuracy*. In traditional LMs, the input is a token sequence without 2D structures. In contrast, GNNs rely on graph topology for each model input and intermediate computation. In addition, current model parallelism methods further split batches to numerous microbatches for higher efficiency, which is non-trivial for GNNs since over-segmentation on graphs loses connectivity and hurts model accuracy substantially.

To address these challenges, we design an Inter-layer Flow Parallelism specialized for LM4Graph methods. Figure 7(c) depicts the workflow with 4 microbatches across 4 devices. Given an L -layer GNN, UniTG partitions the model into P consecutive stages of L/P layers each, assigning one stage per worker. Workers exchange intermediate embeddings (forward pass) and gradients (backward pass) via point-to-point communication at stage boundaries. During the forward pass (blue blocks), the Flow Parallelism first splits the original graph with N nodes into M subgraphs, referred to as *microbatches* hereafter, by considering the affinity nature. This minimizes edge cuts to preserve locality and training quality (more details in §4.4). Then these microbatches are pipelined across P workers with overlapping computation to reduce bubble time. During the backward pass (green blocks), radients are accumulated across microbatches to update stage-local parameters. Notably, aggregation executes on the affinity-aware data layout (yellow blocks).

F-then-B vs. 1F1B. We adopt the Forward-then-Backward (F-then-B) strategy in GPipe [33], instead of One-Forward-One-Backward (1F1B) [17, 23, 51], as *F-then-B better fits the memory and time optimizations when the number of microbatches is small*. As Figure 8 illustrates, increasing the microbatch count significantly degrades GNN convergence and accuracy (orange and red curves). Therefore, the microbatch count for pipelined GNN should be limited to a small scale, which is typically no more than the parallelism degree (i.e., $M \leq P$). In this case, 1F1B offers no throughput advantage as its efficiency gains require a massive number of microbatches [33, 38, 60], whereas F-then-B achieves comparable throughput with more balanced per-worker memory. We therefore employ F-then-B in UniTG, leveraging its pipeline bubbles to interleave LM training (§4.4).

4.3 Dual-modality Collaborative Learning

Existing LM4Graph methods fine-tune pre-trained LMs (e.g., DeBERTa [26]) to better encode node text attributes. Yet they treat

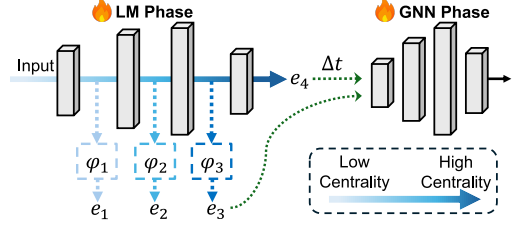


Figure 9: Illustration of the proposed Centrality-guided Lazy Tuner, and how it interacts with the GNN phase. Each grey bar represents a sequence of model layers. Both phases are trained in the collaborative learning mechanism.

LM fine-tuning as a standard NLP task, applying uniform computation across all nodes while ignoring graph topology. This incurs prohibitive overhead: full fine-tuning over the entire node set is computationally expensive and neglects structural properties that differentiate nodes in importance. While parameter-efficient techniques such as adapters [29, 30] and prompt tuning [45, 48] reduce LM training costs, they remain graph-agnostic. Those differences invoke specialized and dedicated system designs for LM4Graph methods towards more aggressive LM fine-tuning and more efficient training paradigms.

Centrality-guided Lazy Tuner. Motivated by the early exiting mechanism [9, 16, 47, 56], we propose Centrality-guided Lazy Tuner that adopts graph-aware early exits during LM fine-tuning. Figure 9 depicts the mechanism. Rather than routing all nodes through the full model depth, we exploit node centrality to determine exit depth. The key insight is that *not every node deserves the same depth of consideration, and one shall opt for fast reaction to marginal nodes without time-consuming overthinking*. Therefore, for graph inputs to the LM, UniTG assigns each node a degree centrality $c(v) = |\mathcal{N}(v)|$, precomputed in $O(|V|+|E|)$ from the adjacency matrix, and partitions all nodes into four distinct groups $\mathcal{G} = \{C_1, C_2, C_3, C_4\}$ by centrality, with C_1 (light blue) containing the least central nodes and C_4 (deep blue) the most central. The LM is also evenly divided into 4 parts as exit positions. During fine-tuning, the training of nodes in C_1 leaves earliest from a quarter of the network to an early-exit linear layer ϕ_1 . Then ϕ_1 converts the hidden states into output embeddings e_1 (similar for other groups). After applying the early exit strategy, the learning objective becomes:

$$\min \frac{1}{|\mathcal{G}|} \sum_{i \in |\mathcal{G}|} \mathcal{L}_i^{\text{exit}}, \quad (3)$$

where $|\mathcal{G}|$ is the number of exits, and $\mathcal{L}_i^{\text{exit}}$ denotes the standard loss function at the i -th exit layer. Prior work on LM early exit [9, 16, 47] prove that the early-exit losses decay at a similar pace to the final-exit loss. Though the early-exit losses might be higher than the final-exit loss, they encode less important nodes that play trivial roles in subsequent training (low centrality). Applying it to UniTG, our Centrality-guided Lazy Tuner can provide robust embeddings for graph learning and obtain convergence comparable to full LM fine-tuning (see Figure 13(b)).

Modality Collaborator. To convey the encoded embeddings from the LM to the GNN phase, current approaches save the embeddings as checkpoints. Then the GNN task is launched to load them as model inputs. This cold-start way is time-consuming and error-prone, which will be exacerbated when phases alternate and repeat

Algorithm 1 Dual-modality Collaborative Training

Input: LM $\mathcal{LM}$, GNN $\mathcal{GNN}$, TAG $G=(V, A, s_V)$, GNN epochs J ; trigger threshold $T=20$, sync schedule $S=\{0.3J, 0.2J, 0.1J, 0.02J\}$

Output: Trained $\mathcal{LM}$ and $\mathcal{GNN}$

```

1: Partition  $V$  into  $\{C_1, \dots, C_4\}$  by degree centrality  $c(v)=|N(v)|$ 
2:  $H[v] \leftarrow$  raw feature of  $v$  for all  $v \in V$   $\triangleright$  initial GNN inputs
3:  $idx \leftarrow 0, \Delta t \leftarrow S[0], cnt \leftarrow 0, F \leftarrow +\infty$ 
4: for  $j = 1$  to  $J$  do
5:   Fine-tune  $\mathcal{LM}$  via Lazy Tuner on  $\{C_1, \dots, C_4\}$ 
6:   if  $j \bmod \lfloor \Delta t \rfloor = 0$  then  $\triangleright$  periodic sync
7:      $H[v] \leftarrow$  exit embedding  $\mathcal{LM}(s_v)$ 
8:      $\mathcal{L}_j^{\text{GNN}} \leftarrow$  train  $\mathcal{GNN}$  one epoch with input  $H$ 
9:      $F_{\text{prev}} \leftarrow F$ 
10:    if  $F = +\infty$  then  $F \leftarrow \mathcal{L}_j^{\text{GNN}}$  else  $F \leftarrow 0.2F + 0.8\mathcal{L}_j^{\text{GNN}}$ 
11:    if  $F < F_{\text{prev}}$  then  $cnt \leftarrow cnt + 1$  else  $cnt \leftarrow 0$ 
12:    if  $cnt \geq T$  and  $idx < |S| - 1$  then  $\triangleright$  tighten sync
13:       $idx \leftarrow idx + 1, \Delta t \leftarrow S[idx], cnt \leftarrow 0$ 
14: return  $\mathcal{LM}, \mathcal{GNN}$ 

```

frequently. Therefore, we introduce a new paradigm for coherent and portable graph learning. We fuse two phases' training simultaneously and coordinate them to gradually enhance the learning. As shown in Figure 9, we initiate both the LM and GNN training at the beginning. During LM fine-tuning, we periodically extract node embeddings from the last (or early-exit) hidden layer every Δt steps and feed them directly to the GNN as input features. Nodes whose embeddings have not yet been synchronized fall back to raw numerical features to preserve training stability.

Dynamic Synchronization Scheduling. To better ensure training accuracy, we dynamically adjust the synchronization frequency Δt between LM and GNN phases. Prior works [15, 84] demonstrate that even pretrained LM embeddings (without fine-tuning) improve GNN accuracy. Based on this, since the embeddings generated in the early tuning stages are insufficiently refined, intuitively we begin with infrequent synchronization and progressively increase the frequency as training stabilizes. Specifically, to measure the overall training status of using LM encoded embeddings, we track an Exponential Moving Average (EMA) Loss: $F_j = 0.8F_{j-1} + 0.2\mathcal{L}_j^{\text{GNN}}$, where $\mathcal{L}_j^{\text{GNN}}$ is the GNN loss at epoch j . Initially, $\Delta t = 0.3J$, where J is the total number of GNN epochs. Then Δt gradually decreases to values in the set $\mathcal{S} = \{0.3J, 0.2J, 0.1J, 0.02J\}$ determined empirically. When F_j keeps decreasing for 20 consecutive epochs, we conclude that the current embeddings are sufficiently refined, and UniTG decreases Δt to the next value in $\mathcal{S}$ to get more frequent synchronization of LM-generated embeddings. Otherwise, the embeddings are not refined enough, and we keep Δt unchanged to enable more stable training. Algorithm 1 shows the details of UniTG's collaborative learning. We note that the 20-epoch condition is evaluated on the smoothed F_j (EMA with a factor of 0.8), not the raw stochastic loss. The heavy smoothing effectively filters out SGD noise, making consecutive decreases a practically achievable condition. The final accuracy in Table 5 indicates the effectiveness of our Modality Collaborator.

Handling Feature Staleness. Collaborative training introduces feature staleness, as the GNN may consume embeddings from prior LM iterations. We argue that this staleness is naturally mitigated by two properties of our design. First, *early-stage tolerance*: at the

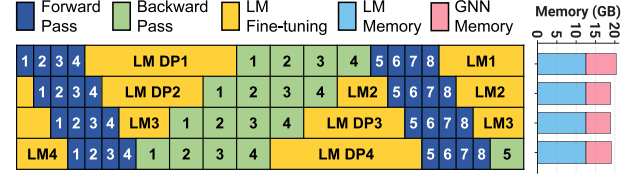


Figure 10: Illustration of Bubble Interleaver with 4 pipeline stages and 4 microbatches. Right-side is the memory consumption of each GPU.

beginning of training, LM embeddings change rapidly, but the GNN is far from convergence, so the impact of stale features on the optimization trajectory is limited. Second, *late-stage stability*: as the LM fine-tuning progresses, the encoded embeddings gradually stabilize, which diminishes staleness. Our adaptive Δt schedule aligns with this observation: it starts with infrequent synchronization (large Δt) when staleness is high but harmless, and increases synchronization frequency as embeddings mature and become more beneficial for GNN training. We provide a sensitivity analysis on Δt in §5.3 to empirically validate this design.

4.4 Streamlined Pipeline Schedule

We then discuss execution-level optimizations, including a Bubble Interleaver to improve resource utilization in a time-sharing way, and an Auto Scheduler tailored for the Dual-modality Collaborative Learning to adjust training settings and support general usage.

Bubble Interleaver. The Flow Parallelism can significantly reduce the GNN training time, but also starve faster stages of microbatches to run and thus result in resource under-utilization. As shown in Figure 7(c), the inter-layer values (activations and gradients) of the partial model are transferred at the forward and backward passes between different workers. For worker 1, after the forward pass of the final microbatch (blue block 4), it keeps waiting since gradients of microbatch 1 have not been produced yet (green block 1), leaving the GPU idle for a long time (other workers are similar).

UniTG designs Bubble Interleaver as shown in Figure 10, which leverages bubbles to greatly extend the tuning job resources in a time-sharing way, almost without hurting the throughput of the pipelined GNN training. LM4Graph workloads are perfectly suitable for the bubble interleaving execution due to the following unique features: (1) *Mixed parallel workloads*. Unlike common learning jobs, there are two distinct distributed jobs (i.e., LM and GNN) that coexist in the learning process, enabling time-sharing computation. (2) *LM throughput insensitivity*. In LM4Graph methods, usually the LM phase consumes a small fraction of total training time (Figure 5) and is further accelerated by our Lazy Tuner, thus being tolerant of partial throughput slowdown. (3) *Reduced resource demand*. Lazy Tuner instructs the LM to exit from earlier layers along the training. Therefore, the resources needed for LM computation gradually reduce and the bubbles can be utilized for more LM tuning. All the above inspires us to leverage the spare resources of the bubbles and execute the LM phase in a pause-and-resume way.

Implementation Details. In Figure 10, UniTG interleaves the 4-stage GNN Flow Parallelism with *data-parallel* LM fine-tuning. Since LMs are relatively small, data parallelism (DP) or single-GPU training is sufficient for efficient LM fine-tuning. We discuss the system implications of employing larger LMs in §7, noting that

their practical utility for LM4Graph workloads remains marginal (Table 1). Here we depict LM tuning with DP across 4 GPUs for illustration, with each DP process (denoted as LM DP1~4) running in FP bubbles accordingly. Each LM process executes in a pause-and-resume mechanism to leverage the bubbles. We register hooks on each operation of the LM to promptly control the pause and resumption in the forward and backward passes of each layer. Initially, both LM and GNN phases are started, and UniTG identifies the idle resources in FP bubbles. For the pipelined GNN, our Affinity-aware Flow Parallelism also registers hooks inside each pipeline stage to monitor its training progress and resource consumption. To interleave the LM process correctly into bubble intervals, Bubble Interleaver checks the status of the CUDA streams of the NCCL kernels. We realize this by sending self-defined signals to control the resumption and suspension of LM processes at the beginning and end of bubbles, which avoids kernel interference and introduces negligible overhead to GNN training. We also profile the memory footprints with Bubble Interleaver (Figure 10, right). The result confirms that flow parallelism and data parallelism reduce per-device memory footprints sufficiently to eliminate swapping during colocation, while substantially improving GPU utilization.

Auto Scheduler. Unlike typical hybrid parallelism methods [4, 24, 86], the parallelisms in our scenario are innately independent, which avoid complicated strategy design and can be better scheduled by UniTG. We design an Auto Scheduler to balance the efficiency-quality trade-off and accommodate various LM4Graph methods to deliver the best performance. Two factors are considered.

(1) *Microbatch count M .* Unlike LLM training, the number of microbatches in graph learning is highly limited, as discussed in §4.2. While increasing M improves throughput, it degrades accuracy due to a greater drop of connectivity information. To better determine M , we profile GNN accuracy by varying M values (Figure 8). The results show that $M \leq 4$ gives ideal performance across all tested configurations, with smaller graphs being more tolerant of more microbatches. Therefore, given the Flow Parallelism degree P and graph size N , empirically we can derive M as follows to balance training efficiency and model accuracy:

$$M = \begin{cases} 16 & N < 10^4 \\ 8 & 10^4 \leq N < 5 \times 10^4 \\ 4 & N \geq 5 \times 10^4 \end{cases} \quad \text{and} \quad M \leq P \quad (4)$$

(2) *Scenarios where UniTG performs best.* The effectiveness of Bubble Interleaver varies depending on multiple factors and we present the most affected ones. (i) *Higher parallelism of both phases.* More stages of a pipelined GNN imply a higher bubble ratio and more spare resources, which typically can interleave the LM tuning on each device and accelerate dozens simultaneously. In this case, the same size of DP for the LM tuning is also a perfect match for maximum resource utilization and overall acceleration. (ii) *Tasks with similar training time are preferred.* Too large LMs of up to 50 billion parameters fail to be allocated on any bubble resources and are memory-consuming in both model and activation size. On the other hand, too small LMs (e.g., 20 million parameters) terminate far earlier than GNN training and are unable to fully utilize the hardware resources. As a result, LMs of middle size like DeBERTa [26] are ideal choices for the LM phase.

Datasets	# Vertices	# Edges	# Features	Task	Metric
ogbn-arxiv [31]	169,343	1,166,243	128	40-class classif.	Accuracy
ogbn-products* [31]	54,025	74,420	100	47-class classif.	Accuracy
ogbl-citation2 [31]	2,927,963	30,561,187	128	Link prediction	MRR
pubmed [59]	19,717	44,338	500	3-class classif.	Accuracy
cora [49]	2,708	5,429	1,433	7-class classif.	Accuracy

Table 3: Detailed information of evaluated datasets.

Models	ogbn-arxiv & ogbn-products			pubmed & cora		
	# Layers	# Hidden	Epoch	# Layers	# Hidden	Epoch
RevGCN	112	224	1000 & 500	448	80	1000 & 1000
RevGAT	20	256	2000 & 500	40	768	1000 & 500
ResGEN	56	128	2000 & 1000	112	64	500 & 1000

Table 4: Hyperparameters for GNNs.

5 EVALUATION

Implementation. We implement UniTG atop PyTorch 2.1 [55] with about 23K LoC. The point-to-point communication is implemented via `torch.distributed` with the NCCL backend. We evaluate UniTG across four dimensions: (1) training throughput, (2) model accuracy, (3) energy consumption, and (4) microbenchmarks and ablation studies isolating the contribution of each component.

Testbed. Our experiments are performed on two NVIDIA GPU servers with the following setups: ❶ 2 servers each with 8 *type A* GPUs (24GB). Intra-server connections (CPU-GPU, GPU-GPU) are based on PCIe 4.0x16 lanes and inter-server connections are via 1Gbps Ethernet. ❷ 2 servers each with 8 *type B* GPUs (96GB) with NVLink and 3.2Tbps InfiniBand. By default, server ❶ is used.

Datasets and Models. We evaluate UniTG on versatile real-world graph datasets with multiple scales (Table 3). Following TAPE [28], we mainly use a subset of ogbn-products for fair comparison (the full ogbn-products dataset is specifically evaluated in §5.1). We use three large GNNs commonly adopted as the GNN backbone, including RevGCN [41], RevGAT [39], and ResGEN [40]. For the LM backbone, we exemplify DeBERTa [26] in evaluations. Note that UniTG can be also applied to other LM4Graph approaches with versatile LM and GNN backbones. Hyperparameters follow the original papers and are summarized in Table 4.

Baseline Implementations. As no prior integrated LM4Graph systems exist, we compare against the original implementations of two representative methods: (1) GLEM [84], which trains the LM and GNN alternatively with many rounds, and (2) TAPE [28], which captures textual information by an LLM and then translates via an LM several times to get different encodings. As both baselines only support a single GNN architecture, we meticulously extend them to additional backbones for comprehensive comparison. We note that both baselines follow a decoupled paradigm where the LM and GNN phases execute sequentially one-at-a-time. For distributed training, we follow their original implementation where both baselines only support data parallelism for the LM phase.

5.1 End-to-end Performance

Table 5 reports end-to-end training time and test accuracy on one GPU-A server (speedup is relative to GLEM). By default, UniTG employs 4-stage GNN flow parallelism and 4-way LM data parallelism. **Learning efficiency.** For node classification tasks, UniTG substantially outperforms all baselines by 4.5~17.3×. The makespan of GLEM can prolong to prohibitive 23 hours with RevGCN on the

Model	Method	ogbn-products		ogbn-arxiv		pubmed		cora	
		Total Time(s)	Test Acc.(%)	Total Time(s)	Test Acc.(%)	Total Time(s)	Test Acc.(%)	Total Time(s)	Test Acc.(%)
RevGCN	GLEM	7593.20(1.00×)	79.36	81945.19(1.00×)	72.52	9604.11(1.00×)	94.59	5943.11(1.00×)	86.66
	TAPE	6289.28(1.21×)	80.38	71694.70(1.14×)	73.45	8604.29(1.12×)	94.37	5767.62(1.03×)	87.64
	UniTG	1453.84(5.22×)	79.34	4732.49(17.32×)	72.37	2123.67(4.52×)	95.06	1278.38(4.65×)	86.53
RevGAT	GLEM	3962.34(1.00×)	79.82	46924.24(1.00×)	74.93	13550.53(1.00×)	94.71	1392.88(1.00×)	87.96
	TAPE	2761.58(1.43×)	79.87	38542.30(1.22×)	76.02	12224.39(1.11×)	94.65	1071.87(1.30×)	87.08
	UniTG	621.83(6.37×)	79.05	5347.00(8.78×)	75.80	1304.79(10.39×)	94.83	277.69(5.02×)	87.82
ResGEN	GLEM	OOM	-	OOM	-	2507.30(1.00×)	94.41	804.89(1.00×)	85.42
	TAPE	OOM	-	OOM	-	1645.64(1.52×)	93.94	316.82(2.54×)	86.35
	UniTG	433.46	76.55	4099.88	72.39	401.04(6.25×)	94.73	92.06(8.74×)	86.72

Table 5: End-to-end evaluation of methods when training on one GPU-A server. OOM means the method runs out of memory.

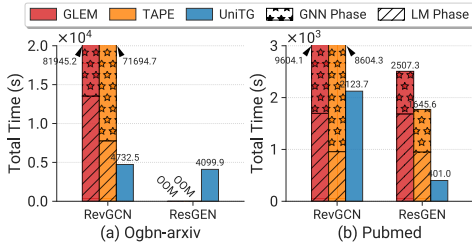


Figure 11: Detailed time breakdown of LM and GNN phases for all methods on two models.

ogbn-arxiv dataset. In contrast, UniTG successfully reduces the makespan to only 1.3 hours, which critically facilitates learning efficiency and saves possible resource costs. Moreover, the speedup of UniTG is more evident on larger datasets with more intensive computation. This reflects that UniTG is more suitable for large-scale jobs with limited resources, which is hard to handle by existing methods. In most cases, TAPE achieves marginal improvement over GLEM, as both rely on sequential phase execution. Additionally, the two baselines run out of memory (OOM) on both ogbn datasets on ResGEN, due to their incapability of training large GNNs.

Figure 11 breaks down training time by phase. In most cases, the GNN phase dominates over 89% of the total time, which hinders system efficiency and blocks resource usage under the decoupled paradigm. Even when GNN is faster than LM (e.g., RevGEN in Figure 11(b)), the total time is still large due to the accumulated time of two phases. In contrast, UniTG directly completes the learning quickly in one step, with a makespan even lower than a single LM phase in baselines (e.g., RevGCN on ogbn-arxiv), thanks to our intra-phase acceleration and inter-phase interleaving.

Accuracy. Table 5 also reports test accuracy. UniTG achieves accuracy comparable to baselines, sometimes slightly below TAPE (which employs LLM-generated explanations for augmentation). Notably, UniTG always achieves accuracy improvement of up to 0.79% on pubmed, mainly thanks to data-aware LM tuning. In summary, UniTG is sufficient to provide equivalent accuracy with other complex baselines.

Power consumption. UniTG designs a time-sharing paradigm, which eliminates the decoupled pattern where each phase monopolizes the hardware resources and causes energy waste. In Table 6, we compare the total energy consumption of UniTG on each GPU against baselines over several datasets. UniTG demonstrates

Method	ogbn-arxiv		pubmed	
	RevGCN	RevGAT	RevGCN	RevGAT
GLEM	27.72M	15.43M	3.04M	4.62M
TAPE	24.49M(1.1×)	12.86M(1.2×)	2.75M(1.1×)	4.20M(1.1×)
UniTG	1.56M(17.8×)	1.65M(9.4×)	0.69M(4.4×)	0.46M(10.0×)

Table 6: GPU energy consumption comparison (units: J).

Method	RevGCN		ResGEN		Method	Time (hours)	MRR (%)
	Time(s)	Acc.(%)	Time(s)	Acc.(%)			
GLEM	98.1	77.56	OOM	-	GLEM	155.8	81.02
TAPE	79.8	78.08	OOM	-	TAPE	126.4	81.85
UniTG	9.3	77.36	3.4	74.86	UniTG	11.5	81.13

(a) ogbn-products.

(b) ogbl-citation2.

Table 7: End-to-end results on full-scale datasets and additional TAG tasks (ogbn-products for node classification and ogbl-citation2 for link prediction).

up to 17.8× and 15.7× energy savings over GLEM and TAPE, respectively. GLEM spends the most GPU power because of its tedious LM and GNN steps for multiple rounds. Our system cleverly fuses two phases’ training, dramatically cutting down unnecessary energy costs and promoting sustainable computing for graphs.

Scalability on the full ogbn-products. To further validate the scalability of UniTG on larger graphs, we conduct experiments on the full ogbn-products dataset (2.4M nodes, 62M edges), as shown in Table 7a. To train the full ogbn-products dataset, we set the number of mini-batches to 46, and keep the number of micro-batches as 4 as original. Both baselines complete training on RevGCN but exhaust GPU memory (OOM) on ResGEN, as they are hard to scale to large GNN models. UniTG successfully trains both backbones: on RevGCN, it achieves 10.5× and 8.6× speedup over GLEM and TAPE (9.3 s vs. 98.1 s and 79.8 s, respectively) with comparable accuracy; on ResGEN, it completes training in 3.4 s at 74.86% accuracy where baselines fail.

Link prediction. To evaluate the generality of UniTG beyond node classification, we conduct link prediction experiments on ogbl-citation2 [31], a large-scale text-attributed citation network (2.9M nodes, 30M edges). Since all GNN backbones used in our work were originally designed for node-level tasks, we adopt the SEAL framework [78] with RevGCN as the subgraph encoder for UniTG and baselines. Table 7b reports the end-to-end training time and Mean Reciprocal Rank (MRR). UniTG achieves 13.5× and 11.0× speedup over GLEM and TAPE, while maintaining comparable MRR (81.13% vs. 81.02% and 81.85%).

Variant	ogbn-arxiv	pubmed
GLEM	81945.19 ($\downarrow 17.32\times$)	9604.11 ($\downarrow 4.52\times$)
Complete UniTG	4732.49 (1.00 $\times$)	2123.67 (1.00 $\times$)
w/o Affinity-aware FP (Runtime)	15007.60 ($\downarrow 3.17\times$)	7064.40 ($\downarrow 3.33\times$)
w/o Bubble Interleaver (Execution)	7470.97 ($\downarrow 1.58\times$)	3070.50 ($\downarrow 1.45\times$)
w/o Modality Collaborator (Algorithm)	4968.12 ($\downarrow 1.02\times$)	2279.41 ($\downarrow 1.07\times$)
w/o Lazy Tuner (Algorithm)	4813.50 ($\downarrow 1.01\times$)	2130.02 ($\downarrow 1.00\times$)

Table 8: Component ablation study. Each row disables one component, and the ratio is relative to the complete UniTG.

Component ablation study. To demonstrate that each component of UniTG is necessary and contributes meaningfully, we conduct a leave-one-out ablation study. Starting from the full UniTG, we disable one component at a time and report the end-to-end training time and test accuracy with RevGCN in Table 8. More detailed ablation studies without affinity reordering and flow parallelism can be found in §5.2 and Figure 12, so here we regard Affinity-aware FP as a single component. And note that if we disable Affinity-aware FP, the Bubble Interleaver cannot be applied either since there are no pipeline bubbles. To compare each module fairly, in this case we colocate the LM phase and the GNN phase to simulate the effect of Bubble Interleaver. From the results, we can clearly observe the synergistic effect of the components of UniTG’s three-layer design. Assume each component contributes independently, on ogbn-arxiv, the expected speedup over GLEM should be $81945.19 / (4732.49 \times 3.17 \times 1.58 \times 1.02 \times 1.01) \approx 3.36\times$, much lower than that of the complete system which is $17.32\times$.

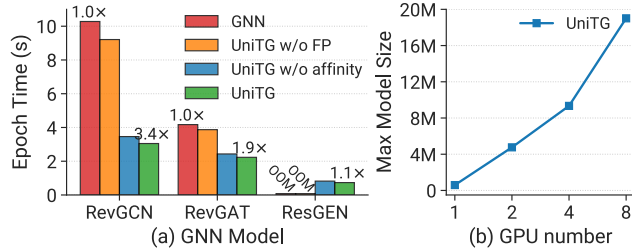


Figure 12: Detailed ablation study of Affinity-aware FP on ogbn-arxiv. (a) Effect of each technique. (b) The supported maximum GNN size w.r.t. GPU count.

5.2 Effect of Flow Parallelism

Training speed w.r.t. different techniques. Figure 12(a) reports the training throughput of the GNN phase when varying the applied techniques. Obviously, the proposed Flow Parallelism (blue bar) reduces the training time by a large margin of $3\times$ on the basis of vanilla GNN training. Applying affinity-aware reordering on top of it (green bar) further shortens the epoch time by $3.4\times$, owing to its effectiveness in reducing irregular memory access. UniTG can support large GNNs like ResGEN which suffers OOM issue on baseline methods. Moreover, Figure 12(a) reveals an interesting observation that UniTG achieves limited improvement over vanilla GNN if FP is disabled. This is because sparse computation dominates in large GNNs and graphs, exhibiting no evident speedup even if we cluster the graph. Therefore, it is important to combine FP and affinity-aware reordering to achieve the desired performance.

Model size w.r.t. number of GPUs. In Figure 12(b), we examine the maximum GNN model size that can be trained on 1~8 GPUs with UniTG. Since all baselines do not support large GNN training

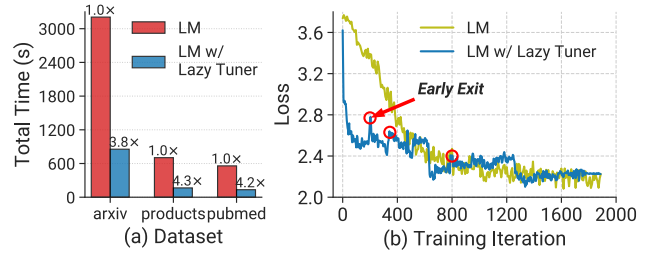


Figure 13: Effect of Lazy Tuner on DeBERTa-base. (a) Total time effect on LM fine-tuning. (b) Loss curve comparisons with some highlighted exit points.

on multiple GPUs, we only plot the results of our system. The maximum GNN size of UniTG almost scales linearly w.r.t. the number of GPUs and can reach to 19 million on 8 GPUs, which is unavailable for baselines employing vanilla training.

5.3 Effect of Collaborative Learning

Centrality-guided Lazy Tuner. We first evaluate the Lazy Tuner on the LM phase (§4.3), measuring tuning time and model quality on one type-A GPU server.

(1) *Efficiency of LM fine-tuning.* Figure 13(a) compares total LM tuning time with and without the Lazy Tuner across three datasets. We report total time rather than per-iteration time because the adaptive exit depth varies across iterations. The Lazy Tuner reduces tuning time by $3.8\times$ - $4.3\times$. For example, on ogbn-products, tuning time drops from 701 s to 165 s. Due to the page limit, we only select DeBERTa-base [26] as the LM backbone for presentation because of its relatively obvious efficacy of Lazy Tuner.

(2) *Loss curves and convergence.* Since Lazy Tuner only trains a partial model for certain inputs, we compare loss trajectories with and without the Lazy Tuner. From the results in Figure 13(b), the loss curve with Lazy Tuner (blue) decays at a faster pace at the beginning, owing to early exits of unimportant nodes from the network. Unsurprisingly, when approaching the end of training, the early-exit losses are slightly higher than the standard loss (olive). Notably, we can observe some spikes in the loss curves (red circles), standing for the occurrence of exiting at a different layer, e.g., 16-th layer other than the original final layer on the first spike. Though there exist many spikes for the early-exit loss during the whole process, it shows an overall convergence trend and the system is tolerant to less precise embeddings for less important nodes.

Model accuracy of Modality Collaborator. We regularly synchronize embeddings from the LM to enhance GNN training. Therefore, we examine the model accuracy and convergence curves of UniTG in Figure 14. UniTG matches TAPE’s convergence trajectory and final accuracy across all configurations, confirming that collaborative learning well preserves the model quality. On ogbn-arxiv with RevGCN, UniTG additionally accelerates convergence, consistent with the faster LM tuning dynamics observed in Figure 13(b). **Sensitivity analysis on synchronization frequency.** To further understand how the synchronization interval Δt affects final accuracy and how UniTG handles potential feature staleness, we conduct a sensitivity analysis by varying Δt with fixed values and comparing against our adaptive schedule. We train RevGAT on ogbn-arxiv

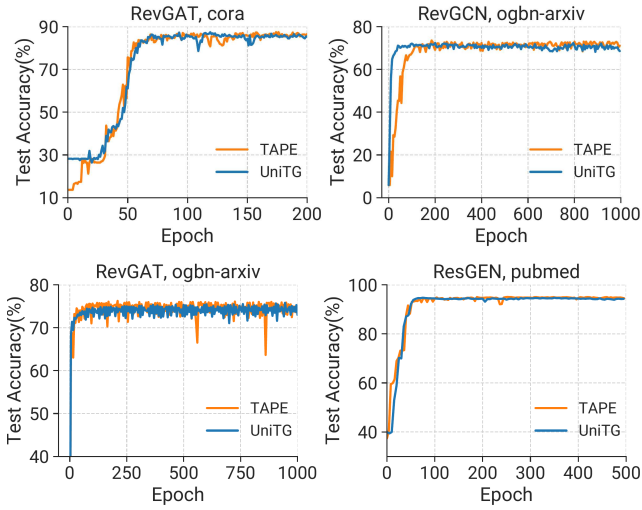


Figure 14: The convergence curve comparisons of UniTG and TAPE on different models and datasets on one type-B server.

Sync. Interval Δt	ogbn-products		ogbn-arxiv	
	Total Time(s)	Test Acc.(%)	Total Time(s)	Test Acc.(%)
0.3J (infrequent)	563.79	68.14	3908.54	65.34
0.1J	572.09	75.07	3935.88	68.23
0.02J (very frequent)	633.77	75.48	4185.38	71.20
Ori (Sync at each step)	684.02	79.04	4304.86	73.08
Adaptive (UniTG)	621.83	78.90	4099.88	72.86

Table 9: Sensitivity analysis on synchronization interval Δt with RevGAT. J is the total GNN training epochs.

Exit Strategy	ogbn-arxiv		pubmed	
	Time(s)	Acc.(%)	Time(s)	Acc.(%)
Centrality-guided (ours)	854.62	72.36	133.38	95.16
Random Exit	855.10	69.45	133.94	94.10
Full-depth (no exit)	3206.49	72.41	557.58	95.21

Table 10: Ablation of early-exit strategy with RevGAT. Random assignment randomly places nodes into exit groups.

and pubmed, and report the final test accuracy in Table 9. Infrequent synchronization (0.3J) minimizes wall-clock time (563.8 s and 3908.5 s) but incurs substantial accuracy loss (68.1% and 65.3%) due to prolonged feature staleness. Per-step synchronization achieves the highest accuracy (79.0% and 73.1%) yet requires the longest training time (684.0 s and 4304.9 s). The adaptive schedule matches per-step accuracy to within 0.2% while reducing training time by 9.1% and 4.8%, respectively. It defers frequent synchronization until the EMA loss tracker confirms embedding quality, then progressively increases the rate. This validates that early-stage staleness is benign while late-stage precision is critical (§4.3).

Centrality-guided vs. random early exiting. To validate that node centrality provides a meaningful signal for the Lazy Tuner, we compare our centrality-guided group assignment against random assignment, where nodes are randomly placed into exit groups regardless of their graph topology. Table 10 shows the results with RevGAT. Both exit strategies achieve similar time savings over full-depth encoding, since both reduce the average number of LM layers traversed. However, centrality-guided assignment preserves higher downstream GNN accuracy, confirming that the topological

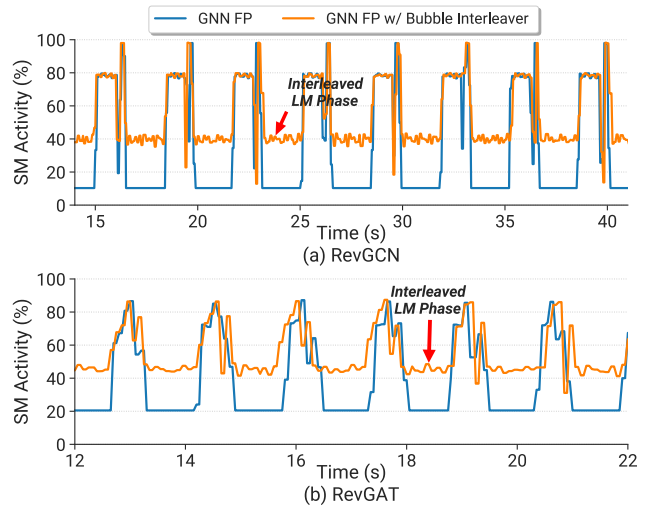


Figure 15: Visualization of GPU utilization via DCGM on (a) RevGCN and (b) RevGAT. Several iterations of the first pipeline stage are presented.

signal (node centrality) meaningfully determines which nodes can tolerate shallow encoding: low-centrality nodes participate less in GNN message passing, so their slightly less refined embeddings have minimal impact on the final model quality.

Convergence analysis. We then intuitively analyze why collaborative learning provides stable convergence to the whole system. The Modality Collaborator delivers embeddings from a progressively fine-tuned LM whose loss decreases as training progresses (Figure 13(b)), producing increasingly refined embeddings over time. The adaptive Δt schedule ensures that synchronization frequency increases *only* when the EMA tracker confirms the embeddings are benefiting GNN training, preventing disruption from noisy early-stage features. As a result, the GNN receives monotonically improving input features throughout training. Combined with the standard convergence properties of SGD on the GNN loss, this yields stable training behavior. The convergence curves in Figure 14 empirically confirm this analysis across all evaluated models and datasets.

5.4 Effect of Streamlined Pipeline Schedule

GPU utilization of Bubble Interleaver. We explore the effects of this module on one type-B GPU server. For better illustration of the impact of Bubble Squeezer, we interleave the LM tuning in data parallelism into the GNN training in FP containing 4 pipeline stages, which is implemented based on PyTorch [55]. To better measure the precise GPU utilization, we employ NVIDIA DCGM [2] to record SM Activity as the utilization metric. We capture the utilization traces on one GPU for 40 seconds with and without Bubble Interleaver on two models in Figure 15. Two traces are collected separately and we align them manually according to the time axis.

Taking Figure 15(a) as an example, for the GNN training only with FP (blue), since the only active kernel in the bubble is the NCCL kernel for communication, we observe the extremely poor SM activity of about 7% during the bubble. On the contrary, UniTG inserts each DP process of the LM phase into the bubbles. Hence, it utilizes the spare SMs, accomplishing a higher SM utilization at

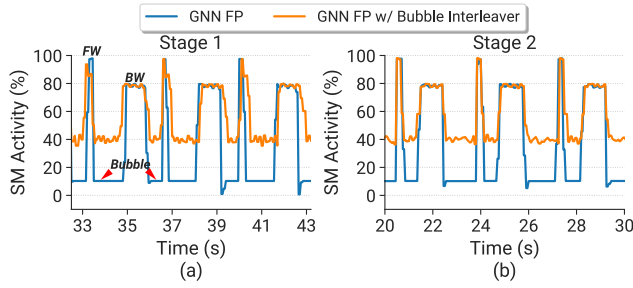


Figure 16: GPU utilization on the second and third stage when training RevGcn in FP.

about 40%. In Figure 15(b), the SM activity is increased to around 45% in bubbles as well. Moreover, in both figures, the bubble-interleaved curve matches well at the computation intervals (where SM activity > 80%) and bubble intervals, with no evident delay compared to the GNN FP curve. This validates that Bubble Interleaver can greatly improve hardware utilization with negligible interference on the original GNN task. We also test the throughput influence of directly collocating two phases and find it causes unacceptable interference (about 40% slowdown for both models).

To prove that each LM DP process is correctly interleaved and executed, we also investigate the GPU utilization on other GPUs, namely other pipeline stages of GNN FP. As shown in Figure 16, the taller spike at almost 100% SM activity stands for the forward pass and the backward pass reaches nearly 80% SM activity. For stage 1 in Figure 16(a), a long bubble occurs right after the forward pass and a short bubble after the backward pass. We can observe the LM DP process on this GPU executes exactly in these two bubble intervals, raising the SM activity to about 40%, and meanwhile not affecting the GNN FP process. The same phenomenon can be seen in the overlapped curves on stage 2. Both validate UniTG can utilize the idle resources timely and accurately. In summary, UniTG successfully achieves much higher GPU utilization during bubble time, and substantially improves training efficiency in a time-sharing way owing to the superior capability of exploiting intermittent GPU resources.

6 RELATED WORK

Language Model for Graph Learning. LM4Graph has recently attracted increasing attention in graph representation learning, particularly for node classification on text-attributed graphs (TAGs) [82]. As illustrated in Figure 2(a), existing LM4Graph approaches generally follow a two-phase paradigm: an LM phase, which leverages pre-trained language models to encode textual node attributes into embeddings, and a GNN phase, which performs message passing over the graph structure using the generated node representations.

According to how the two phases are organized, prior work can be broadly categorized into three groups. The first line of work adopts a cascaded architecture, where the LM and GNN are trained sequentially. Representative methods such as [11, 15, 28, 32, 57, 65, 74, 81, 88] first employ a pre-trained LM to encode node texts and store the resulting embeddings as checkpoints. These fixed embeddings are then used as inputs for a downstream GNN training process. While straightforward, this design fully decouples the LM and GNN, preventing mutual adaptation during training. The

second category of methods alternates between the LM and GNN phases, allowing the output of one phase to enhance the other. Works such as [72, 77, 84] iteratively update the LM and GNN, enabling limited cross-phase interaction. Although this alternating strategy improves representation quality compared to purely cascaded designs, the two phases remain trained in a one-at-a-time manner, leading to frequent phase switching and increased system overhead. Finally, several approaches [14, 73, 76, 80] rely solely on language models to learn node representations and make predictions, without explicitly incorporating graph neural networks. As these methods do not perform structural message passing, they fall outside the scope of this work.

Pipeline Parallelism Methods. Pipeline parallelism has been extensively studied as an effective technique to improve GPU utilization in model-parallel distributed training. GPipe [33] and PipeDream [52] are among the first works to introduce pipelined execution for deep neural networks. GPipe partitions a mini-batch into multiple micro-batches and executes them in a pipelined fashion across model stages, thereby overlapping computation and communication. PipeDream further improves pipeline efficiency by allowing asynchronous weight updates and reducing pipeline stalls.

Subsequent studies have proposed more advanced pipeline parallelism techniques to tackle practical system challenges, including reducing pipeline bubbles [43, 52, 61], lowering memory consumption [53], and improving overall system efficiency [17]. These approaches have demonstrated strong scalability for large non-GNN models, particularly Transformers, where training samples within a mini-batch are largely independent. In addition, several recent works have explored pipeline-based execution for GNN workloads [19, 46, 67], aiming to adapt pipelining techniques to graph-structured computation.

7 CONCLUSION

This work presents UniTG, an algorithm-system co-optimized framework tailored for LM4Graph, a state-of-the-art LM-based graph learning method. The primary goal of UniTG is to minimize the makespan of LM4Graph training, while simultaneously improving resource utilization, reducing energy cost, and preserving model quality. To this end, UniTG incorporates three specialized modules that exploit the unique properties of LM4Graph and jointly deliver high performance. Evaluations show UniTG can boost training by up to 17.3× while achieving comparable final accuracy.

Limitations and Future Work. The current design of UniTG is optimized for moderate-size LMs (up to ~560M parameters), which are sufficient for TAG tasks as shown in Table 1. When scaling to very large LMs (e.g., billions of parameters), the Bubble Interleaver may become ineffective because the per-step LM computation time can exceed the GNN pipeline bubble duration, saturating the available idle resources. Potential mitigations such as LM-side model parallelism across bubble slots or activation offloading are promising directions for future work.

ACKNOWLEDGMENTS

This work was supported by the the Fundamental Research Funds for the Central Universities (SWU-KR24043) and the gift from Sichuan JiuXin Technologies Co., Ltd.

REFERENCES

- [1] 2000. Graph partitioning models for parallel computing. *Parallel Comput.* 26, 12 (2000), 1519–1534. [https://doi.org/10.1016/S0167-8191\(00\)00048-X](https://doi.org/10.1016/S0167-8191(00)00048-X) Graph Partitioning and Parallel Computing.
- [2] 2023. NVIDIA Data Center GPU Manager. <https://developer.nvidia.com/dcgmm>.
- [3] Iro Armeni, Sasha Sax, Amir R. Zamir, and Silvio Savarese. 2017. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *CoRR* abs/1702.01105 (2017). <https://arxiv.org/abs/1702.01105>
- [4] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. 2022. Varuna: scalable, low-cost training of massive deep learning models. In *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys '22)*. Association for Computing Machinery.
- [5] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst. 1994. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Society for Industrial and Applied Mathematics. <https://doi.org/10.1137/1.9781611971538>
- [6] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. 2019. Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences* 116, 32 (2019), 15849–15854. <https://doi.org/10.1073/pnas.1903070116>
- [7] Jingji Chen, Zhuoming Chen, and Xuehai Qian. 2023. GNNPipe: Scaling Deep GNN Training with Pipelined Model Parallelism. *CoRR* abs/2308.10087 (2023). <https://arxiv.org/abs/2308.10087>
- [8] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML '20)*. 1725–1735.
- [9] Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024. EE-LLM: large-scale training and inference of early-exit large language models with 3D parallelism. In *Proceedings of the 41st International Conference on Machine Learning (ICML '24)*. JMLR.org.
- [10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*.
- [11] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olga Milenkovic, and Inderjit S Dhillon. 2022. Node Feature Extraction by Self-Supervised Multi-scale Neighborhood Prediction. In *International Conference on Learning Representations (ICLR '22)*.
- [12] Eli Chien, Jianhao Peng, Pan Li, and Olga Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *International Conference on Learning Representations (ICLR '21)*.
- [13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. <https://doi.org/10.18653/v1/N19-1423>
- [14] Tu Anh Dinh, Jeroen den Boef, Joran Cornelisse, and Paul Groth. 2022. E2EG: End-to-End Node Classification Using Graph Topology and Text-based Node Attributes. *CoRR* abs/2208.04609 (2022). <https://arxiv.org/abs/2208.04609>
- [15] Keyu Duan, Qian Liu, Tat-Seng Chua, Shuicheng Yan, Wei Tsang Ooi, Qizhe Xie, and Junxian He. 2023. SimTeG: A Frustratingly Simple Approach Improves Textual Graph Learning. *CoRR* abs/2308.02565 (2023). <https://arxiv.org/abs/2308.02565>
- [16] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020. Depth-Adaptive Transformer. In *International Conference on Learning Representations (ICLR '20)*.
- [17] Shiqing Fan, Yi Rong, Chen Meng, Zongyan Cao, Siyu Wang, Zhen Zheng, Chuan Wu, Guoping Long, Jun Yang, Lixue Xia, Lansong Diao, Xiaoyong Liu, and Wei Lin. 2021. DAPPLE: a pipelined data parallel approach for training large models. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '21)*. Association for Computing Machinery.
- [18] Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3 (2010), 75–174. <https://doi.org/10.1016/j.physrep.2009.11.002>
- [19] Zhenbo Fu, Xin Ai, Qiang Wang, Yanfeng Zhang, Shizhan Lu, Chaoyi Chen, Chunyu Cao, Hao Yuan, Zhewei Wei, Yu Gu, Yingyou Wen, and Ge Yu. 2025. NeutronTask: Scalable and Efficient Multi-GPU GNN Training with Task Parallelism. *Proceedings of the VLDB Endowment* 18 (2025), 1705–1719.
- [20] Karypis George and Kumar Vipin. 1997. METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices.
- [21] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural message passing for Quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML '17)*. JMLR.org.
- [22] Will Hamilton, Zitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems (NeurIPS '17)*.
- [23] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. PipeDream: Fast and Efficient Pipeline Parallel DNN Training. *CoRR* abs/1806.03377 (2018). <https://arxiv.org/abs/1806.03377>
- [24] Aaron Harlap, Alexey Tumanov, Andrew Chung, Gregory R. Ganger, and Phillip B. Gibbons. 2017. Proteus: agile ML elasticity through tiered reliability in dynamic resource markets. In *Proceedings of the Twelfth European Conference on Computer Systems (EuroSys '17)*. Association for Computing Machinery.
- [25] Zellig Harris. 1985. Distributional structure. In *The philosophy of linguistics*.
- [26] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION. In *International Conference on Learning Representations (ICLR '21)*.
- [27] Ruining He and Julian McAuley. 2016. 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 (WWW '16)*.
- [28] Xiaoxin He, Xavier Bresson, Thomas Laurent, Adam Perold, Yann LeCun, and Bryan Hooi. 2024. Harnessing Explanations: LLM-to-LM Interpreter for Enhanced Text-Attributed Graph Representation Learning. In *International Conference on Learning Representations (ICLR '24)*.
- [29] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning (ICML '19)*. 2790–2799.
- [30] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR '22)*.
- [31] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems (NeurIPS '20)*.
- [32] Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai-Wei Chang, and Yizhou Sun. 2020. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '20)*. Association for Computing Machinery.
- [33] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems (NeurIPS '19)*.
- [34] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam Rajbhandari, and Yuxiong He. 2023. DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models. *CoRR* abs/2309.14509 (2023).
- [35] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. In *Proceedings of Machine Learning and Systems (MLSys '20)*.
- [36] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR '16)*.
- [37] Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing Activation Recomputation in Large Transformer Models. In *Proceedings of Machine Learning and Systems (MLSys '23)*.
- [38] Fanxin Li, Shixiong Zhao, Yuhao Qing, Xusheng Chen, Xiuxian Guan, Sen Wang, Gong Zhang, and Heming Cui. 2023. Fold3D: Rethinking and Parallelizing Computational and Communicational Tasks in the Training of Large DNN Models. *IEEE Transactions on Parallel and Distributed Systems* 34, 5 (2023), 1432–1449. <https://doi.org/10.1109/TPDS.2023.3247883>
- [39] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021. Training Graph Neural Networks with 1000 Layers. In *Proceedings of the 38th International Conference on Machine Learning (ICML '21)*. 6437–6449.
- [40] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs Go As Deep As CNNs?. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (CVPR '19)*.
- [41] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. DeeperGCN: All You Need to Train Deeper GCNs. *CoRR* abs/2006.07739 (2020).
- [42] Pengcheng Li, Yixin Guo, Yingwei Luo, Xiaolin Wang, Zhenlin Wang, and Xu Liu. 2022. Graph neural networks based memory inefficiency detection using selective sampling. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '22)*. IEEE Press.
- [43] Shigang Li and Torsten Hoefler. 2021. Chimera: efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. Association for Computing Machinery.
- [44] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. 2023. 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)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 2391–2404. <https://doi.org/10.18653/v1/2023.acl-long.134>
- [45] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 4582–4597. <https://doi.org/10.18653/v1/2021.acl-long.353>
- [46] Zhiyuan Li, Xun Jian, Yue Wang, Yingxia Shao, and Lei Chen. 2024. DAHA: Accelerating GNN Training with Data and Hardware Aware Execution Planning. *Proceedings of the VLDB Endowment* 17 (2024), 1364–1376.
- [47] Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: a Self-distilling BERT with Adaptive Inference Time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 6035–6044. <https://doi.org/10.18653/v1/2020.acl-main.537>
- [48] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. GPT Understands, Too. *CoRR* abs/2103.10385 (2023). <https://arxiv.org/abs/2103.10385>
- [49] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the Construction of Internet Portals with Machine Learning. In *Information Retrieval* 3, 127–163.
- [50] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems (NeurIPS '13)*.
- [51] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*. Association for Computing Machinery.
- [52] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*. Association for Computing Machinery.
- [53] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-Efficient Pipeline-Parallel DNN Training. In *Proceedings of the 38th International Conference on Machine Learning (ICML '21)*. 7937–7947.
- [54] M. E. J. Newman. 2013. Spectral methods for community detection and graph partitioning. *Phys. Rev. E* 88 (Oct 2013), 042822. Issue 4. <https://doi.org/10.1103/PhysRevE.88.042822>
- [55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems (NeurIPS '19)*.
- [56] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently Scaling Transformer Inference. In *Proceedings of Machine Learning and Systems (MLSys '23)*.
- [57] Yijian Qin, Xin Wang, Ziwei Zhang, and Wenwu Zhu. 2023. Disentangled Representation Learning with Large Language Models for Text-Attributed Graphs. *CoRR* abs/2310.18152 (2023). <https://arxiv.org/abs/2310.18152>
- [58] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations (ICLR '20)*.
- [59] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '08)*.
- [60] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).
- [61] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR* abs/1909.08053 (2020). <https://arxiv.org/abs/1909.08053>
- [62] Pierre Stock, Benjamin Graham, Rémi Gribonval, and Hervé Jégou. 2019. Equinormalization of Neural Networks. In *International Conference on Learning Representations (ICLR '19)*.
- [63] Qingxiao Sun, Yi Liu, Hailong Yang, Ruizhe Zhang, Ming Dun, Mingzhen Li, Xiaoyan Liu, Wencong Xiao, Yong Li, Zhongzhi Luan, and Depei Qian. 2022. CoGNN: efficient scheduling for concurrent GNN training on GPUs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '22)*. IEEE Press.
- [64] Shyam A. Tailor, Javier Fernandez-Marques, and Nicholas D. Lane. 2021. Degree-Quant: Quantization-Aware Training for Graph Neural Networks. *CoRR* abs/2008.05000 (2021).
- [65] Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V. Chawla, and Panpan Xu. 2024. Graph Neural Prompting with Large Language Models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '24)*.
- [66] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR '18)*.
- [67] Qiang Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen, Xiaodong Zhang, and Ge Yu. 2022. NeutronStar: Distributed GNN Training with Hybrid Dependency Management. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22)*. Association for Computing Machinery.
- [68] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. 2021. GNNAdvisor: An Adaptive and Efficient Runtime System for GNN Acceleration on GPUs. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI '21)*. USENIX Association, 515–531.
- [69] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.* 38 (2019), 1–12.
- [70] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. *CoRR* abs/1806.03536 (2018).
- [71] Junhan Yang, Zheng Liu, Shitao Xiao, Chaozhao Li, Defu Lian, Sanjay Agrawal, Amit Singh, Guangzhong Sun, and Xing Xie. 2021-12-06. GraphFormers: GNN-nested Transformers for Representation Learning on Textual Graph. In *Advances in Neural Information Processing Systems (NeurIPS '21)*.
- [72] Michihiro Yasunaga, Antoine Bosselut, Hongyu Ren, Xikun Zhang, Christopher D Manning, Percy S. Liang, and Jure Leskovec. 2022-12-06. Deep Bidirectional Language-Knowledge Graph Pretraining. In *Advances in Neural Information Processing Systems (NeurIPS '22)*.
- [73] Michihiro Yasunaga, Jure Leskovec, and Percy Liang. 2022. LinkBERT: Pre-training Language Models with Document Links. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 8003–8016. <https://doi.org/10.18653/v1/2022.acl-long.551>
- [74] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering. *CoRR* abs/2104.06378 (2021). <https://arxiv.org/abs/2104.06378>
- [75] Michihiro Yasunaga, Rui Zhang, Kshitij Meelu, Ayush Pareek, Krishnan Srinivasan, and Dragomir Radev. 2017. Graph-based Neural Multi-Document Summarization. *CoRR* abs/1706.06681 (2017). <https://arxiv.org/abs/1706.06681>
- [76] Ruosong Ye, Caiqi Zhang, Runhui Wang, Shuyuan Xu, and Yongfeng Zhang. 2024. Language is All a Graph Needs. *EACL* (2024).
- [77] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021-12-06. Do Transformers Really Perform Badly for Graph Representation?. In *Advances in Neural Information Processing Systems (NeurIPS '21)*.
- [78] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. Curran Associates Inc.
- [79] Meng Zhang, Qinghao Hu, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2024. Sylvie: 3D-adaptive and Universal System for Large-scale Graph Neural Network Training. In *2024 IEEE 38th International Conference on Data Engineering (ICDE)*.
- [80] Meng Zhang, Jie Sun, Qinghao Hu, Peng Sun, Zeke Wang, Yonggang Wen, and Tianwei Zhang. 2024. TorchGT: A Holistic System for Large-Scale Graph Transformer Training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '24)*. IEEE Press.
- [81] Xikun Zhang, Antoine Bosselut, Michihiro Yasunaga, Hongyu Ren, Percy Liang, Christopher D Manning, and Jure Leskovec. 2022. GreaseLM: Graph REASONing Enhanced Language Models. In *International Conference on Learning Representations (ICLR '22)*.
- [82] Ziwei Zhang, Haoyang Li, Zeyang Zhang, Yijian Qin, Xin Wang, and Wenwu Zhu. 2023. Graph Meets LLMs: Towards Large Graph Models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems GLFrontiers Workshop (NIPS'23 GLFrontiers Workshop)*.
- [83] Haisha Zhao, San Li, Jiaheng Wang, Chunbao Zhou, Jue Wang, Zhikuan Xin, Shunde Li, Zhiqiang Liang, Zhijie Pan, Fang Liu, Yan Zeng, Yangang Wang, and Xuebin Chi. 2025. Acc-SpMM: Accelerating General-purpose Sparse Matrix-Matrix Multiplication with GPU Tensor Cores. In *Proceedings of the 30th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '25)*. Association for Computing Machinery.
- [84] Jianan Zhao, Meng Qu, Chaozhao Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. 2022. Learning on Large-scale Text-attributed Graphs via Variational Inference. In *International Conference on Learning Representations (ICLR '22)*.

- [85] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proceedings of the VLDB Endowment* 16 (2023), 3848–3860.
- [86] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*. USENIX Association, 559–578.
- [87] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. 2020. Towards Deeper Graph Neural Networks with Differentiable Group Normalization. In *Advances in Neural Information Processing Systems (NeurIPS '20)*.
- [88] Jason Zhu, Yanling Cui, Yuming Liu, Hao Sun, Xue Li, Markus Pelger, Tianqi Yang, Liangjie Zhang, Ruofei Zhang, and Huasha Zhao. 2021. TextGNN: Improving Text Encoder via Graph Neural Network in Sponsored Search. In *Proceedings of the Web Conference 2021 (WWW '21)*. Association for Computing Machinery.