SYLVIE: 3D-adaptive and Universal System for Large-scale Graph Neural Network Training

Meng Zhang[1,2,3] Qinghao Hu[1,2,3] Cheng Wan[4] Haozhao Wang[2] Peng Sun[3,5]
Yonggang Wen[2] Tianwei Zhang[2*]
[1]S-Lab, Nanyang Technological University [2]NTU [3]Shanghai AI Laboratory
[4]Georgia Institute of Technology [5]SenseTime

***Abstract*—Distributed full-graph training of Graph Neural Networks (GNNs) has been widely adopted to learn large-scale graphs. While recent system advancements can improve the training throughput of GNNs, their practical adoption is limited by the potential accuracy decline. This concern is particularly prominent in deeper and more intricate GNN architectures, where noticeable performance degradation becomes apparent. Moreover, existing works fail to comprehensively consider diverse opportunities for acceleration. Motivated by these deficiencies, we propose SYLVIE, a full-graph training system that not only improves the training throughput substantially but also maintains the model quality for universal GNNs. By harnessing the inherent information embedded in the graph data and model structure, SYLVIE intelligently optimizes GNN training across three key dimensions: *data*, *time*, and *execution*. It identifies performance-relevant features of the input graph offline as subsequent optimization guidance. Subsequently, SYLVIE devises an online convergence-maintenance strategy that adaptively integrates and aligns GNN-specific quantization and inter-epoch asynchronous training with the real-time training characteristics. Extensive experiments demonstrate that SYLVIE surpasses existing GNN training systems by up to 17.2× speedup for both shallow and deep GNNs, without compromising the model accuracy.**

***Index Terms*—distributed systems, graph neural networks, model-agnostic, 3D-adaptive, accuracy maintenance**

I. INTRODUCTION

In recent years, GNNs have become very popular and exhibited state-of-the-art performance in learning structured data like graphs. Driven by such breakthroughs, GNNs have been applied to a variety of tasks such as community detection [1], [2], node classification [3]–[6], link prediction [7] and time series prediction [8], [9]. GNNs capture the underlying dependencies of the given graph via message passing operations [10], [11]. Despite their impressive performance on graph-related tasks, training GNNs on large-scale graphs containing millions to billions of nodes is still a long-standing issue, as extensive memory resources are needed for loading and computing input graphs [12]–[14] and the memory demand easily exceeds the memory capacity. This hinders the practical development of complex graph datasets and GNN models.

Existing solutions to this problem can be categorized into two directions. First, some works [15]–[19] utilize *sampling-based training*, which only selects a subset of nodes and edges to be trained at each iteration. Although this method can reduce memory consumption, it requires careful consideration of the sampling strategy and may lead to the loss of important neighborhood information, suffering from model accuracy loss [20]–[22]. Second, distributed *full-graph training* [22]–[27] allows training over full graphs on multiple devices or servers to reduce the computing time and memory demand on each GPU. Therefore, it preserves the complete input graph information so that model accuracy can be better preserved. Due to its promising features, our focus in this work lies in developing full-graph training systems.

Existing GNN training systems still meet some deficiencies in practice. First, **they may compromise the model quality and fail to support universal GNN models**. As aforementioned, though sampling-based methods [15], [16], [20], [28]–[30] can reduce the memory footprint, they will forfeit crucial neighborhood information, ultimately resulting in obvious model accuracy degradation ($> 2\%$) compared with full-graph training as shown in Table I. On the other hand, the fast development of GNN algorithms raises urgent demand for the compatibility of the underlying training systems. Shallow GNNs limit the ability to extract high-order neighboring information, especially on large graphs, as mentioned by [31]–[33]. Nowadays more and more deep GNNs emerge, and they are capable of learning representations from larger receptive fields and achieving better accuracy. However, state-of-the-art full-graph training systems [22], [24], [26], [27], [34] only consider specific model architecture (i.e., Graph Convolutional Network (GCN) [14]) with shallow layers (two or three) [22], [26], [35]. They fail to accommodate more advanced GNNs with complex aggregators (e.g., LSTM and attention networks [7], [36]) or deeper GNNs such as DAGNN [31]. Consequently, this limitation of current works leads to model accuracy decline and restricts their applications in practice.

Second, **existing systems overlook the joint optimization opportunities tailored to GNNs**. Generally, GNN training is often hampered by substantial communication and memory bottlenecks, in contrast to DNNs whose computation tends to be the bottleneck (§II-B). Regrettably, prevalent frameworks (e.g., DGL [37] and PyTorch Geometric [38]) do not provide effective distributed full-graph training support. To this end, some works [21], [24], [39], [40] have made efforts to improve the situation. However, they still bear significant communication overhead. Furthermore, existing systems [22], [27], [28], [39], [41] focus on system-level optimizations while ignoring the exploitation of graph data information. Some works like [42] also utilize the input graph to enhance the optimization decisions, but they focus on different input information and

*Corresponding author.

TABLE I: Test accuracy of training GraphSAGE on the Ogbn-products dataset with different sample sizes.

Sample Size	Sampling-based			Full-Graph
	5	10	15	
Accuracy (%)	73.55	74.87	76.84	**79.19**

only cope with single-GPU training on small-scale graphs. They typically target a single optimization aspect and fail to improve training efficiency while preserving model accuracy under the distributed setting.

To bridge these gaps, we design SYLVIE, a novel distributed full-graph GNN training system that supports shallow (e.g., GCN), deep (e.g., DAGNN), and special aggregator (e.g., GAT) GNNs. It jointly optimizes training across three granularities, including *data*, *time*, and *execution* aspects. The core design of SYLVIE comes from the following three insights. First, *the information from GNN inputs guides the system optimizations*. Present GNN models are diverse in layer sequences, aggregation methods, and depths. Similarly, the input graphs vary in node properties and features. By profiling and analyzing these two, we find that valid optimization suggestions tailored to specific GNN tasks can be obtained from the input information. Second, *adaptive optimizations by monitoring the training process can preserve the quality of universal GNN models*. By monitoring the online training process, we enable the training acceleration of deeper GNNs while greatly preserving model quality. We also observe that convergence can be further improved by adopting different optimization choices along the training process. Third, *the benefits of an advanced execution mode (i.e., asynchronous pipeline) can be maximized via curtailing communication*. In the original case, as the model size increases and cluster size scales, the communication overhead in distributed learning dominates the training time. The communication is frequent and heavy while the bandwidth of network interfaces is limited, which significantly diminishes the benefits of pipelining. However, pipelining the reduced communication can manifest its advantages and greatly contribute to efficiency.

Integrating the above insights, we build a model-agnostic GNN training system SYLVIE, consisting of an offline stage and an online stage to facilitate training while improving the model quality. Through extensive experiments, we show SYLVIE substantially outperforms SOTA baselines by up to 17.2× speedup across various models without hurting their accuracy. Such superiority is attributed to two innovative features in SYLVIE. (1) By integrating the online training characteristics, we are the ***first*** to accelerate full-graph training of deeper GNNs on large graphs while preserving the model quality. In contrast, current systems only support limited GNN models with just 2 or 3 layers [24], [26], [27], [43]–[45], failing to accommodate more advanced GNNs with complex aggregators or deeper GNNs such as DAGNN. This can be validated from §VI, Table VII. The evaluated baselines such as BNS-GCN [22] and PipeGCN [26] either lack support on deeper GNNs or suffer from substantial accuracy loss. SAR [40] and PipeGCN [26] also show inferior training speed in most cases. (2) We also propose to exploit graph-level properties to dynamically adjust system optimizations on large-scale graphs. Current GNN systems [22], [26], [41], [44], [46] adopt a monotonous scheme and fail to maximize the training efficiency for specific graph workload settings. Some works like GNNAdvisor [42] also attempt to use input-level information, but they are only tailored for very small graph training while having no support for large-scale graph training.

In short, SYLVIE differs from all existing works in that it is **model-agnostic** and **exploits input-level information on full-graph training**. It optimizes arbitrary GNN training from **data, time, and execution** dimensions. In particular, SYLVIE pioneers in: (1) model-agnostic, (2) dynamic per-node quantization, and (3) adaptive pipeline. In sum, we make the following contributions:

- SYLVIE stands out as the *first* system designed to accelerate universal GNNs in practice. Supported by both theoretical proofs and extensive experiments, SYLVIE can improve training on versatile GNN models, even on deeper GNNs.
- SYLVIE is the *first* to explore the input-level properties (§IV) on expediting and guaranteeing large-scale full-graph training. Besides, we pioneer in identifying the unique opportunities of quantizing communicated messages in GNNs.
- We coordinate a set of system optimizations to substantially facilitate the training efficiency (up to 17.2×) while preserving the model quality in a *3D-adaptive* manner, including a novel *data-* and *time-adaptive* quantization algorithm (§V-A) and an *execution-adaptive* scheme (§V-B).

II. BACKGROUND AND RELATED WORK

A. Graph Neural Networks

A GNN model first aggregates the feature vectors from the nodes' neighbors and then combines them together, which is called message passing [47]. In general, the iterative learning process contains two important steps in each layer: feature aggregation and update. Intuitively, consider a graph $\mathcal{G} = (V, E)$ with an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{|V|\times|V|}$, nodes $V = \{v_1, \cdots, v_{|V|}\}$, edges $E = \{e_1, \cdots, e_{|E|}\}$, and a node feature matrix $\boldsymbol{X} \in \mathbb{R}^{|V|\times d}$. For an arbitrary layer $l \in [1, L]$, the aggregation and update steps can be expressed as:

$$\boldsymbol{z}_v^{(l)} = \rho^{(l)}\left(\left\{\boldsymbol{h}_u^{(l-1)} \mid u \in \mathcal{N}(v)\right\}\right) \tag{1}$$

$$\boldsymbol{h}_v^{(l)} = \phi^{(l)}\left(\boldsymbol{z}_v^{(l)}, \boldsymbol{h}_v^{(l-1)}\right) \tag{2}$$

where $\mathcal{N}(v)$ means the neighboring nodes of node v. The aggregation function $\rho^{(l)}$ takes the embeddings of neighboring nodes $\boldsymbol{h}_u^{(l-1)}$ to get an intermediate aggregated result $\boldsymbol{z}_v^{(l)}$, which then serves as the input to the update function $\phi^{(l)}$ together with the feature embedding $\boldsymbol{h}_v^{(l-1)}$ of node v itself to obtain the learned embedding $\boldsymbol{h}_v^{(l)}$, a column vector of $\boldsymbol{H}^{(l)}$, which is the matrix consisting of all nodes' embeddings at the l-th layer. Different GNNs vary in their aggregation and update functions. In our work, we classify GNN architectures into three types: shallow, deep, and special GNNs. For each

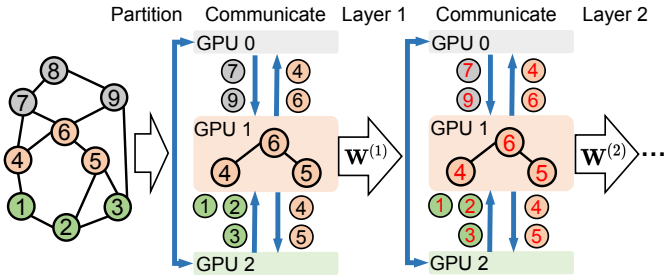

Fig. 1: Example of vanilla distributed GNN training. For the partition on GPU-1, node 4 requires extra features of node 7 from GPU-0 and node 1 from GPU-2 to update its embedding in each layer.

kind, we list one example of the update rule as below: GCN [14], DAGNN [31], and GAT [7].

- Graph Convolution Network (GCN):

$$\boldsymbol{z}_v^{(l)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{d_v d_u}} \mathbf{W}^l \boldsymbol{h}_u^{(l-1)}, \quad \boldsymbol{h}_v^{(l)} = \sigma\left(\boldsymbol{z}_v^{(l)}\right)$$

  where d_v refers to the degree of node v, σ is an activation function, and $\mathbf{W}^l$ is the weight matrix at layer l.

- Deep Adaptive Graph Neural Network (DAGNN):

$$\boldsymbol{Z} = MLP(\boldsymbol{X})$$
$$\boldsymbol{H}^{(L)} = stack(\boldsymbol{Z}, \widehat{\boldsymbol{A}}^1\boldsymbol{Z}, \widehat{\boldsymbol{A}}^2\boldsymbol{Z}, ..., \widehat{\boldsymbol{A}}^L\boldsymbol{Z})$$

  where $\widehat{\boldsymbol{A}} = \widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}$, $\widetilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$. $\widetilde{\boldsymbol{D}}_{v,v} = \sum_u \widetilde{\boldsymbol{A}}_{v,u}$ is the diagonal node degree matrix, $\boldsymbol{I}$ is the identity matrix.

- Graph Attention Network (GAT):

$$\boldsymbol{z}_v^{(l)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \alpha_{v,u} \mathbf{W}^l \boldsymbol{h}_u^{(l-1)}, \quad \boldsymbol{h}_v^{(l)} = \sigma\left(\boldsymbol{z}_v^{(l)}\right)$$

  where α represents the attention coefficients.

B. Distributed GNN Training

Figure 1 shows the vanilla distributed GNN training on full graphs. The whole input graph is first partitioned via a graph partitioning algorithm (e.g., METIS [48]) on the host side to fit into a single GPU. Since each node and its features will only be assigned to one GPU, there exist nodes that are connected to the local partition but reside on other partitions, dubbed **boundary nodes**. For instance, node 4 requires the embeddings of boundary node 7 from GPU-0 and node 1 from GPU-2 to update itself during message passing. In the backward pass, the embedding gradients of boundary nodes are also transferred. Therefore, both embeddings and embedding gradients of boundary nodes, denoted as *messages*, will be transferred in each layer. This communication overhead is non-trivial since the amount of boundary messages can be excessive, as shown in Table II.

Unlike classical distributed DNN training [49], [50] where training samples are independent of each other, it is non-trivial to apply data parallelism to GNNs due to the node dependency between partitions, which will lead to obligatory data communication overhead. To show the massive communication cost more intuitively, we profile the distributed GNN training epoch time and its breakdown in some cases, as shown in Figure 2. We can see for all cases, the communication time nearly dominates the entire training process (up to 89.23%), while the computation and the transfer of weight gradients (all-reduce) only occupy a very small portion. This is also different from distributed DNN training where the transfer of weight gradients (all-reduce) is most costly. The scalability and efficiency of distributed GNN training thus are seriously restrained due to this excessive communication overhead.

TABLE II: Data volume of communicated messages (embeddings & embedding gradients) and weight gradients of three models on the Ogbn-products dataset over 4 GPUs. 4-GCN-256 means a 4-layer GCN with a hidden size of 256.

Model	Embeddings	Embedding Gradients	Total Messages	Weight Gradients
4-GraphSAGE-128	1.56 GB	1.55 GB	3.11 GB	0.40 MB
4-GCN-256	3.10 GB	3.10 GB	6.20 GB	0.65 MB
3-GAT-256	2.07 GB	2.07 GB	4.14 GB	0.41 MB

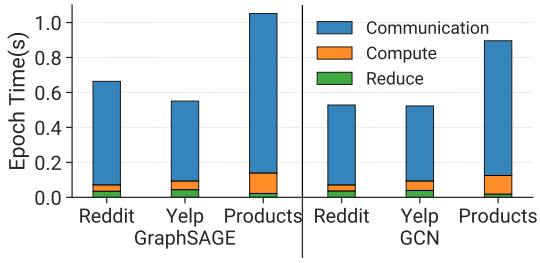

Fig. 2: Training time per epoch in vanilla distributed training with DGL on a single server (8 GPUs).

Prior works propose new frameworks to accelerate distributed GNN training [51], e.g., AliGraph [25] and NeuGraph [41]. However, these methods all store the partitions in CPUs, which inevitably incur frequent CPU-GPU swapping and largely impair the benefits of distributed training. DistDGL [24] provides the scaling results but only on sampling-based methods. LLCG [52] totally drops dependent information between partitions and adds a global correction server to compensate for the error with redundant overhead. Moreover, those works only support mini-batch training on graphs rather than full-graph training. Different from the above sampling-based works, ROC [21] accelerates distributed full-graph training, but it also stores the partitions in CPUs with huge CPU-GPU data transfer cost. Some works [39], [40], [43] improve the performance of full-graph training at scale, but suffer from extra computation burden due to the complexity of introduced operations. BNS-GCN [22] adopts random sampling on the boundary nodes and shows impressive acceleration, yet it risks downgrading the model performance by dropping node connections and its performance is highly dependent on the graph structure.

TABLE III: Comparison of prior works on GNN quantization.

Features	EC-Graph [53]	BiFeat [54]	Degree-Quant [55]	SYLVIE
Distributed Environment	✔	✔	✔	✔
GPU Support	✘	✔	✘	✔
Full-Graph Support	✔	✘	✔	✔
Message Quantization	✔	✔	✘	✔
Adaptive Configuration	✔	✘	✘	✔
Deep GNN Support	✘	✘	✘	✔

C. Quantization for GNNs

Quantization has already been applied in DNNs to accelerate inference [56], [57] or to compress activations to reduce memory consumption [58]. Different from these works, we aim to speed up the distributed GNN training by **quantizing the communication**. This has been studied to be explored for gradient compression in distributed DNN training [59]–[62]. However, the bottleneck of large DNNs stems from the transfer of **weight gradients**. On the contrary, GNNs have a much smaller size of weight gradients, while their layer-wise exchange of **embeddings** and **embedding gradients** is the main bottleneck. To better illustrate this, we train three representative GNN models in a distributed way and record the transferred volume of messages and weight gradients in Table II. Obviously, the size of weight gradients in the GNN case is far smaller than that of transferred embeddings and embedding gradients. Hence, compression methods in distributed DNN training cannot be simply grafted to our scenario.

In recent years, there also emerge various works which apply the quantization technique on GNNs. Model quantization [63], [64] via simulation for memory reduction is a common direction, with the underlying computation still occurring in the 32-bit full precision. EXACT [13] aims to reduce memory demand at the cost of extra training time overhead, seriously deteriorating the training efficiency. Other works like [65] quantize GNN models for efficient inference. EC-Graph [53] also optimizes distributed GNN training by quantizing the communication but only for CPU clusters and empirically adjusts the quantization configuration. Degree-Quant [55] quantizes GNN models and parameters on small graphs, but the training efficiency on GPU clusters even downgrades. BiFeat [54] mainly targets mini-batch training and suffers from non-negligible accuracy loss. Table III summarizes the main differences between SYLVIE and some GNN quantization works. Compared with our work, all these methods have different targets or only consider small-scale datasets. More importantly, none of them considers the generality of deeper or special GNNs.

Challenge of GNN Message Quantization. Quantization of the interacted messages can greatly reduce communication time. As shown in Figure 3, the communication overhead decreases rapidly with the decrease in bit-widths. In particular, 1-bit quantization cuts down almost 89.8% of communication overhead and 84.2% of training time per epoch compared to the full-precision case. However, it also deteriorates the accuracy. Particularly, lower bit-widths come with more accuracy reduction. To further demonstrate this, we showcase the

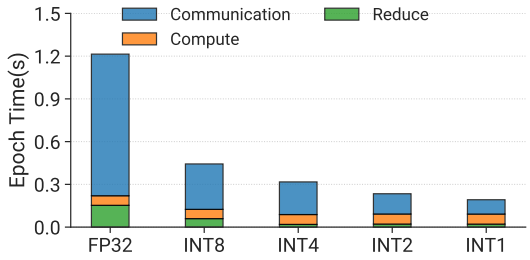

Fig. 3: Training time per epoch and its breakdown when using different quantization bit-widths to train GraphSAGE on Yelp over 8 GPUs.

TABLE IV: The test accuracy (%) of three models trained with different quantization bit-widths. 4-GCN denotes a GCN with 4 layers and the other models are similar.

Dataset	Model	FP32	INT8	INT4	INT2	INT1
Amazon	4-GraphSAGE	81.29	81.09	79.14	79.12	79.09
Amazon	4-GCN	53.7	53.59	53.53	53.34	53.16
Reddit	8-JKNet	92.75	92.66	92.73	91.99	90.91

experiment results over various models and datasets in Table IV. All experiments are done under a fixed number of epochs which is sufficient for all models to converge. The number of epochs is set to 2000 for GraphSAGE and GCN, and 800 for JKNet. It is apparent that the smaller the bit-widths, the greater the reduction in accuracy. To maximize the benefits brought by quantization without sacrificing the model quality, it is necessary to dynamically adapt the quantization level.

D. Pipeline of Distributed GNN Training

While quantization can significantly reduce the communication overhead, it cannot completely eliminate the communication latency. Pipelining the layer-wise communication with computation [26] shows the potential to fully hide the communication time. Different from synchronous training, the model directly begins each layer's computation with the stale messages obtained from the previous epoch, with the communication proceeding between partitions concurrently. The communication currently being overlapped is for the use of the next epoch, ensuring the data integrity when the computation starts.

Challenge of GNN Pipeline Training. The efficiency benefits of simply pipelining computation and communication are limited in GNN training. When the cluster size scales and layer size increases, the communication overhead dominates the training time and the efficacy of pipelining will corrupt badly, as shown by the results in §VI-A. Existing works [27] utilize historical messages via cache to improve the pipelining efficiency, but come with increased training error when the number of stale epochs is large. Considering these limitations, we propose to jointly exploit the quantization and pipeline strategy, which inherits both benefits including bounded staleness control and minimized communication latency.

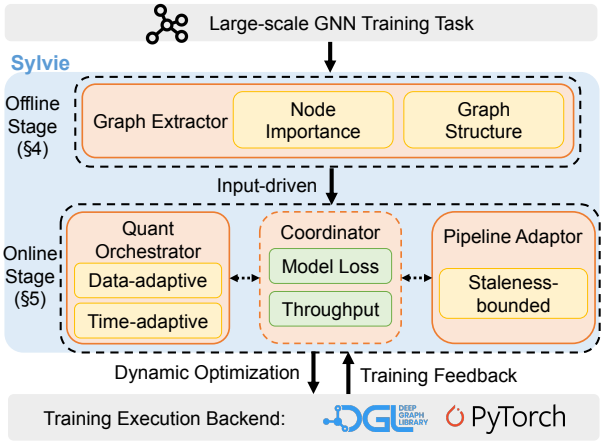

Fig. 4: Overview of SYLVIE architecture and workflow.

III. System Overview

To achieve efficient distributed GNN training while maintaining model accuracy, we design SYLVIE as depicted in Figure 4. It realizes the dynamic optimization via two key stages: **offline stage** for graph property profiling and **online stage** for improving the training efficiency and model performance. Specifically, the offline stage contains one key module:

- *Graph Extractor*: exploits the input-level graph information for potential performance benefits and quantization suggestions in guiding the system-level optimizations.

The online stage contains three key modules:

- *Quant Orchestrator*: dynamically orchestrates the quantization of communicated messages from both data- and time-adaptive perspectives.
- *Pipeline Adaptor*: adjusts the training process between the synchronous and asynchronous modes in a staleness-bounded way.
- *Coordinator*: deploys the 3D-joint optimization decisions and keeps monitoring the training feedback, where DGL [37] and PyTorch [66] serve as the backend.

We elaborate on the details in the following §IV and §V.

GNN Training with SYLVIE. Here we illustrate the distributed GNN training process on the full graph with SYLVIE in Figure 5. The graph is first partitioned into several subgraphs and allocated to different GPUs or servers. In each partition, the inner node set (orange circles) as well as the boundary node set are constructed for the preparation of later message exchange. During both the forward and backward passes of each layer, *Quant Orchestrator* first adaptively quantizes messages sent to other partitions into low-bit integers (❶). Then those quantized data are transferred to the corresponding partitions through network communications (❷). Upon arrival at other partitions, these quantized messages are dequantized back to full-precision values for subsequent computation (❸). Then *Coordinator* deploys the joint optimizations on GNN model training and continuously monitors the feedback to improve training (❹). SYLVIE successfully balances the trade-off between training efficiency and model quality in a *3D*-adaptive manner including the *data* dimension, *time* dimension, and *execution* dimension.

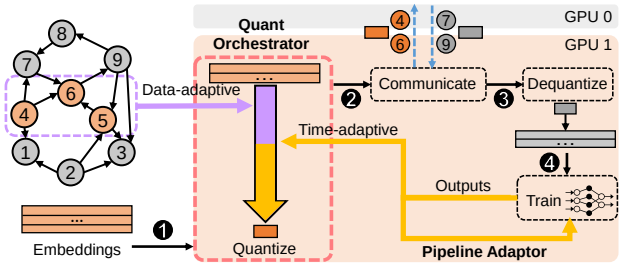

Fig. 5: The training process with SYLVIE. Orange circles and rectangles represent nodes allocated to GPU-1 and their corresponding messages. The others in gray represent nodes/messages on other GPUs.

IV. Offline Stage of Sylvie

In this section, we show the input graph information can guide the system optimization based on our key observation that nodes with different in-degrees will favor different optimization decisions.

Quantization of SYLVIE. Different from prior works in traditional DNNs that quantize all the activations [13], [58] or models [55], [63], [67], we propose quantizing only the exchanged messages to reduce the communication cost in distributed GNN training via the stochastic integer quantization strategy [58]. Specifically, at the forward pass of each l-th GNN layer, each GPU quantizes the embedding of each boundary node $\boldsymbol{h}^{(l)} \in \boldsymbol{H}^{(l)}$ to b-bit integers:

$$\hat{\boldsymbol{h}}_b^{(l)} = \left\lfloor \frac{\boldsymbol{h}^{(l)} - \min(\boldsymbol{h}^{(l)})}{scale} \right\rceil \tag{3}$$

where $\hat{\boldsymbol{h}}_b^{(l)}$ is a node embedding quantized to b-bit at the l-th layer, $scale = \frac{(\max(\boldsymbol{h}^{(l)}) - \min(\boldsymbol{h}^{(l)}))}{2^b - 1}$ is the scaling factor, $\min(\boldsymbol{h}^{(l)})$ is the zero-point, and $\lfloor \cdot \rceil$ is the stochastic rounding operation [68]. Before conducting layer computation, each GPU receives the quantized embeddings $\hat{\boldsymbol{h}}_b^{(l)}$ from the other GPUs and dequantizes them back to 32-bit floating-point values $\tilde{\boldsymbol{h}}^{(l)}$:

$$\tilde{\boldsymbol{h}}^{(l)} = scale \cdot \hat{\boldsymbol{h}}_b^{(l)} + \min(\boldsymbol{h}^{(l)}) \tag{4}$$

Sources of Errors. Many real-world graphs follow the power-law distribution [69] of node degrees. Such distribution leads to some nodes having a substantially larger number of neighbors than others (e.g., large node degree). In the aggregation phase, a node updates itself by pulling messages from its neighbors that send information towards it. This is the main source of substantial numerical errors. Hence, the errors of a node become more significant as its **in-degree** increases. We take partition on GPU-1 in Figure 1 as an example. Though only node 4 and 6 from partition 1 contribute to the in-degree of boundary node 7, in the process of information flowing, node 8 will get messages flowing from partition 1, thus finally passing these messages to node 7. Therefore, for more accurate message passing in subsequent hops, we utilize the **global in-degree value** in the whole graph rather than local in-degree relative to connected partitions to represent importance later. Here we recapitulate the relation between quantization error

and node in-degree following [55]. Taking the GCN layer as an example, the error of neighbor aggregation is $\boldsymbol{y}_v = \sum_{u\in\mathcal{N}(v)\cup\{v\}} \frac{1}{d_v d_u}\left(\tilde{\boldsymbol{h}}_u^{(l)} - \boldsymbol{h}_u^{(l)}\right)$. We can trivially derive that $\mathbb{E}\left(\boldsymbol{y}_v\right) = \mathcal{O}(\sqrt{d})$. The variance of the aggregation output is also $\mathcal{O}(\sqrt{d})$ when the network converges without over-fitting $\sum_{i\neq j}\mathrm{Cov}\left(X_i, X_j\right) \ll \sum_i \mathrm{Var}\left(X_i\right)$.

This verifies that the mean and variance of the aggregation output values increase as node in-degree increases for most GCN-based GNNs. For other GNN models, similar conclusions can also be summarized from Figure 3 in [55]. Further, the quantization error in aggregation also introduces errors in subsequent weights. Through the update rule in GCN, we can get the weight gradients:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{v\in V}\sum_{v:u\in\mathcal{N}(v)} \frac{1}{\sqrt{d_v d_u}}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_v^{(l+1)}} \circ \sigma'\left(\boldsymbol{z}_v\right)\right) {\boldsymbol{h}_u^{(l)}}^\top$$

It is obvious that the larger aggregation error in ${\boldsymbol{h}_u^{(l)}}^\top$, the larger error in the weight gradients, resulting in model quality degradation. Also, from this equation we can see the errors introduced by quantization will accumulate along the number of layers, suggesting that the introduced noise is limited for shallow GNNs. Therefore, existing systems that introduce some errors (e.g., [22], [26]) perform well on shallow GNNs while failing on deeper ones. To address this issue, we design an online adjustment scheme in a 3D way to prevent overmuch errors and encourage more accurate messages to flow back to weights in layers, thus surpassing existing works on deeper GNNs. The dequantization process has a variance term $\mathrm{Var}(\tilde{\boldsymbol{h}}^{(l)}) = \frac{D\cdot scale^2}{6}$ (D is the dimension of hidden layers), even though the expected dequantized message is unbiased $\mathbb{E}\left[\tilde{\boldsymbol{h}}^{(l)}\right] = \mathbb{E}[\mathrm{Deq}(\mathrm{Q}(\boldsymbol{h}^{(l)}))] = \boldsymbol{h}^{(l)}$. Therefore, to address the introduced aggregation error, intuitively we can apply node-aware quantization to improve weight update accuracy.

A. Graph Extractor

To deploy optimizations aiming at specific GNN settings, *Graph Extractor* learns and analyzes the graph structure and node properties for dynamically adjusting the quantization level of each node. This means that even within a single round, each boundary node can be assigned a different quantization bit-width b. *Graph Extractor* first collects the structure information of input graphs and analyzes the importance of each node, then allocates higher bit-width quantization for more important nodes. In this way, SYLVIE can encourage more accurate embeddings and gradients by protecting important nodes from excessive quantization.

Data-adaptive Quantization. To encourage more accurate embeddings and weight updates, SYLVIE protects boundary vertices with higher in-degree values while unprotected vertices operate at reduced precision, since the in-degree value describes the vertex's importance in the boundary vertex pool. Empirically, we also find that high in-degree nodes contribute most towards errors in weight updates. For undirected graphs, we use degree values to define importance as the out-degree value equals the in-degree value. Specifically, before training, we pre-process the input graph and construct an importance factor $p(0 \sim 1)$ for each **boundary node** to denote its probability of introducing quantization errors to embeddings and gradients. A higher importance factor means a higher probability of causing errors. The importance factor is higher if the node's in-degree is large and nodes with the same in-degree also have the same importance. Boundary nodes with the maximum in-degree values are assigned to $p = 1$ and the important factors of other nodes are calculated by interpolating between 0 and 1 based on their in-degree ranking in the boundary nodes pool. We re-order the tensor of in-degree values and match them with evenly distributed percentiles. After this, we generate an importance-aware node mask to map nodes to their corresponding quantization bit-width. This node mask is later combined with the time-adaptive part (illustrated in §V-A) to jointly decide the assigned bit-width for boundary nodes. Nodes with the same mask level are quantized in the same bit-width for communication of both embeddings and embedding gradients. In this way, SYLVIE lowers communication overhead while encouraging more accurate messages to flow back to weights via high in-degree boundary nodes.

V. ONLINE STAGE OF SYLVIE

After exploiting the input-level information, SYLVIE further monitors the training status on the fly and makes optimizations dynamically tailored to GNN training.

A. Quant Orchestrator

SYLVIE explores the opportunity of quantization to reduce the substantial communication in GNNs for training acceleration. It integrates a novel *Quant Orchestrator* to balance the efficiency-accuracy trade-off for distributed GNN training on GPUs. It is computationally lightweight and effective in boosting training and empirically keeping the model quality. In detail, it jointly orchestrates the quantization from two dimensions to minimize communication, namely data and time dimensions. The first dimension lies in the graph's node level as discussed in §IV-A. The second dimension lies in the GNN training time, where we identify that different training epochs can use different quantization bit-widths to reach the efficiency-convergence balance.

Convergence versus Bit-width. From Equation 3 and the variance term $\mathrm{Var}(\tilde{\boldsymbol{h}}^{(l)})$, we can see that changing the bit-width b leads to a trade-off between the total communication volume and the variance value. With smaller b, the communication cost decreases while the variance increases. Building on this, we empirically analyze the effect of b on the model convergence over time and use it to design a strategy to accustom b during the training course. The motivation behind the adoption of time-adaptive quantization during training to minimize communication can be understood from Figure 6. We can see from Figure 6(a) that a smaller b, i.e., coarser quantization, results in worse convergence of training loss versus training time. We also plot the training loss with respect to the communication volume in Figure 6(b), where C is the

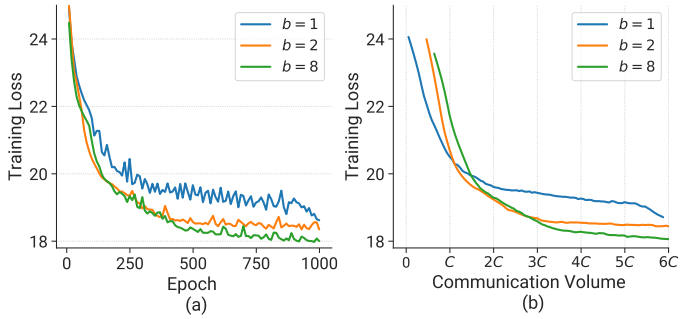

Fig. 6: Training loss with respect to different quantization bit-widths on GAT (Yelp dataset).

unit communication volume we use to plot loss values and equals 5GB here. It reveals that a smaller b enables to perform more epochs for the same communication volume and achieves higher convergence speed in early training stages, which is also pointed out in [70] on a similar problem. Based on this observation, intuitively we can start from the smallest bit-width b along the time dimension and dynamically adjust it according to the training status to balance the training efficiency and convergence. Next, we will introduce the designed metrics to formalize the optimization problem.

Time-adaptive Quantization. Intuitively, time-adaptive quantization changes the quantization bit-width b_t along epoch t during training. *Quant Orchestrator* chooses b_t to minimize the communication overhead without sacrificing the model accuracy as much as possible. Some works [70], [71] use similar observations but only monotonically increase the quantization level. Differently, *Quant Orchestrator* monitors both the training loss (to measure the model convergence) and throughput (to measure the training efficiency) to adjust b_t in a nonmonotonic way, which boosts the training as much as possible while not sacrificing the model convergence significantly. At the end of epoch t, *Coordinator* takes the training outputs, and the global loss $\mathcal{L}_t$ of N partitions is estimated using the local losses ($\mathcal{L}_t^n$): $\mathcal{L}_t = \frac{\sum_{n=1}^{N} \mathcal{L}_t^n}{N}$. To better estimate the convergence, we track a running average loss $F_t = \lambda F_{t-1} + (1-\lambda)\mathcal{L}_t$. To integrate the consideration of training throughput, a Loss Descent Rate (LDR) tailored to GNN training is measured as $LDR_t = \frac{F_t - F_{t-1}}{et_t}$, where et_t is the t-th epoch training time.

According to the aforementioned analysis, at the beginning b_0 is initialized as b_{min} from the bit-width set $\{1, 2, 4, 8\}$. When $LDR_t \geq LDR_{t-\delta}$ for some $\delta \in \mathbb{N}$ which specifies the range of epochs for LDR comparisons, SYLVIE heuristically determines the current bit-width suffices to reduce the loss. Then *Quant Orchestrator* interacts with *Coordinator* to decrease b to gain higher speed. On the condition of this well-balanced training, *Quant Orchestrator* adapts $b_{t+1} = \frac{b_t}{2}$ for higher throughputs. On the other hand, $LDR_t < LDR_{t-\delta}$ in δ epochs denotes the training is about to converge or too many errors are introduced by quantization. The partly trained model requires higher precision to get further improved. In this case, *Quant Orchestrator* increases the bit-width $b_{t+1} = 2b_t$ to reach lower errors and enable more stable and accurate training. The

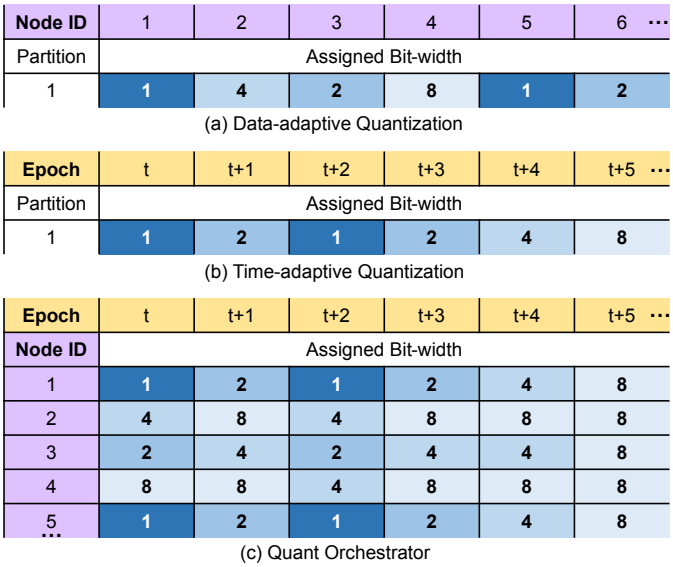

Fig. 7: The process of how *Quant Orchestrator* jointly orchestrates the quantization from time- and data-dimension.

above time-adaptive quantization process is summarized as the following equation:

$$b_{t+1} = \begin{cases} b_{\min} & t = 0 \\ 2b_t & LDR_t < LDR_{t-\delta} \ \ \& \ \ t > \delta \ \ \& \ \ b_t < b_{\max} \\ b_t/2 & LDR_t \geq LDR_{t-\delta} \ \ \& \ \ t > \delta \ \ \& \ \ b_t > b_{\min} \\ b_t & \text{else} \end{cases} \tag{5}$$

The time-adaptive quantization on communication messages ensures models sensitive to noise quickly reach a sufficiently high bit-width and those not sensitive to noise get accelerated as much as possible. It empirically achieves a good trade-off between training convergence and efficiency.

Joint Orchestration. As shown in Figure 5, *Quant Orchestrator* combines the data-adaptive (§IV-A) and time-adaptive quantization to meticulously facilitate training. As illustrated previously, the data-adaptive part constructs an importance-aware node mask in the offline stage. At each epoch t during training, the time-adaptive counterpart first determines a base bit-width b_t. Then the data-adaptive counterpart uses the node mask to further adjust the node-wise bit-widths $b_t^v, v \in V_{\text{boundary}}$ on the basis of b_t in a targeted manner. Figure 7 gives an example of the detailed process of how the time-adaptive part, data-adaptive part and *Quant Orchestrator* adjust the bit-widths. In Figure 7(a), for partition-1, the data-adaptive quantization assigns small bit-widths (e.g., $b = 1$) to less important nodes (1 and 5) and large bit-widths ($b = 8$) to more important nodes. On the other hand, time-adaptive quantization in Figure 7(b) alters the bit-widths for all nodes across training epochs, e.g., it increases from 1 to 2 at epoch $t + 1$. In Figure 7(c), *Quant Orchestrator* applies the data-adaptive counterpart on the basis of time-varying bit-widths. For instance, at t-epoch the time-adaptive counterpart determines a preliminary $b_t = 1$, then the candidate node-wise bit-widths are $\{1, 2, 4, 8\}$. However, the base bit-width is

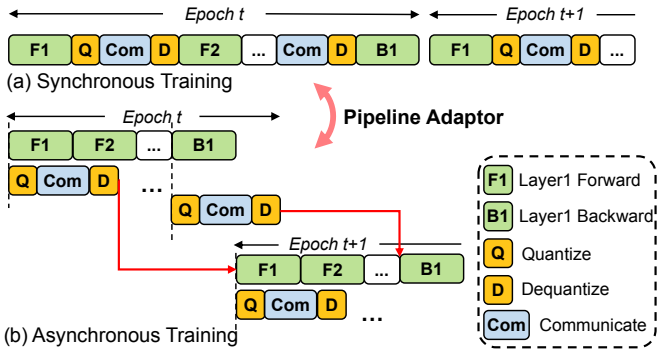

Fig. 8: Illustration of how *Pipeline Adaptor* adjusts execution mode between synchronous and asynchronous training.

8 at epoch $t+5$, so all nodes will be assigned with $b_t^v = 8$ regardless of their importance.

B. Pipeline Adaptor

In the former parts, *Quant Orchestrator* enhances the efficient training of GNNs by reducing the communication volume from two dimensions. However, there exist some large-scale distributed GNN training jobs, where communication still occupies a large portion of the training time. Additionally, the asynchronous training [61], [62], [72] is usually adopted in distributed DNN training to enhance the algorithm efficiency. However, some frameworks like [62] are based on a centralized compute topology with workers running asynchronously to hide partial communication of **weights** and **weight gradients** to the parameter server, suffering from completely stale weight gradients. Pipe-SGD [61] proposes a decentralized learning framework pipelining the local training iterations to hide the communication of **weight gradients**. Nonetheless, all these works target large models, where the main communication overhead comes from the communication of **weights/weight gradients** other than the **embeddings/embedding gradients** in distributed GNN training (as introduced in §II-C). Moreover, different from the staleness of **all** weights/weight gradients in asynchronous distributed DNN training, the staleness in our case incurs only in **partial** embeddings/embedding gradients.

In the online stage, SYLVIE designs *Pipeline Adaptor* which leverages the pipeline of layer-wise communication and computation across two adjacent epochs to further hide all the latency (quantization/dequantization operations and reduced communication duration). As shown by Figure 7 in [26], asynchronization inevitably leads to stale messages, and the errors of staleness will also accumulate in deeper layers. The gradient and feature errors of the second layer are almost twice of the first layer. Therefore, most existing works adopting pipelining perform only well on shallow GNNs. The errors explode and convergence is corrupted when they are deployed on deeper GNNs. In contrast, our *Pipeline Adaptor* automatically adapts training between the synchronous and asynchronous settings to bound the number of delay epochs for staleness control in each layer as shown in Figure 8, enabling SYLVIE on deeper GNNs. The asynchronous pipeline is perfectly suitable for our case due to one unique feature: the quantization and dequantization operations perform simple linear mappings to message vectors, which are low-overhead and thus can be easily parallelized. To better illustrate how *Pipeline Adaptor* works, we first introduce the vanilla synchronous training in Figure 8(a). After each layer's computation, the intermediate activations of boundary nodes are quantized and transferred during both forward and backward passes between all workers using all-to-all communication [22]. The subsequent computation cannot begin until the worker receives and dequantizes the messages. Thus each worker is blocked from computation and cannot continuously utilize the GPU.

In Figure 8(b), to realize the inter-epoch pipeline, each layer's computation directly begins with the latest updated messages in this worker. In parallel, messages are quantized and communicated concurrently. To realize the asynchronous training, we wrap the GPU kernels of inner nodes' computation and pipelined operations (quantization, dequantization and communication) with independent CUDA streams. Note that the communicated boundary messages at epoch t will be used for computation at epoch $t+1$, leading to a compound usage of the latest inner nodes' messages and stale boundary nodes' messages. To mitigate the effects on the convergence of the partial staleness, *Pipeline Adaptor* performs compulsory synchronization of the latest messages for staleness control, reaching a good trade-off between the training throughput and convergence rate. To achieve this, *Pipeline Adaptor* also monitors LDR introduced in §V-A to evaluate the training status. In detail, it determines convergence is downgraded by staleness when $LDR_t < LDR_{t-\delta}$ and informs *Coordinator* to perform synchronous training at epoch $t+1$. Otherwise, the training stays in the asynchronous mode.

C. Coordinator

In the online stage, *Coordinator* retrieves and analyzes the training outputs. It interacts with *Quant Orchestrator* and *Pipeline Adaptor* to coordinate the optimizations on training jointly. Past works [13], [58] prove the convergence of GNNs with quantization as long as the quantization is unbiased and has bounded variance, which has been claimed in §II-C. In addition, SYLVIE only applies quantization to partial messages (messages of boundary nodes), which also limits the introduced variance. Similar methods can be found in existing works [55], [73], [74] which adopt the subset quantization. In addition, [26] demonstrates the convergence of distributed GNN training under the asynchronous setting and the convergence rate is even better than sampling-based methods. These convergence results can extend to SYLVIE and we refer to the detailed analysis from them.

VI. EVALUATION

We implement SYLVIE atop DGL 0.9 [37] and PyTorch 1.10 [66]. The communication process is implemented via `torch.distributed` in the ring all2all pattern [22]. For graph partitioning, we use the widely-adopted METIS [48]

TABLE V: Detailed information of datasets used in evaluation.

Datasets	# Nodes	# Edges	Features Dim.	# Classes
Reddit	232,965	114,615,892	602	41
Yelp	716,847	6,977,410	300	100
Ogbn-products	2,449,029	61,859,140	100	47
Amazon	1,598,960	132,169,734	200	107
Ogbn-papers100M	111,059,956	1,615,685,872	128	172

TABLE VI: Model architecture and detailed hyperparameters.

Model	Config	Dataset			
		Reddit	Yelp	Ogbn-products	Amazon
GraphSAGE	*Arch.*	4 ×256	4×512	3×128	4×128
	HP.	(2000, 0.5)	(2000, 0.1)	(500, 0.3)	(2000, 0.1)
GCN	*Arch.*	4×256	4×512	3×128	4×128
	HP.	(2000, 0.5)	(2000, 0.1)	(500, 0.3)	(2000, 0.1)
GCNII	*Arch.*	8×256	8×512	8×128	6×128
	HP.	(1000, 0.5)	(1000, 0.5)	(500, 0.5)	(2000, 0.5)
DAGNN	*Arch.*	8×256	-	8×128	6×256
	HP.	(1000, 0.8)		(500, 0.8)	(1000, 0.5)
SGC	*Arch.*	8×256	8×512	8×128	6×256
	HP.	(1000, 0.1)	(1000, 0)	(500, 0)	(500, 0)
GAT	*Arch.*	2×256	2×256	3×128	3×128
	HP.	(200, 0.5)	(1000, 0.1)	(200, 0.3)	(1000, 0.1)

Arch.: Number of layers × Number of hidden neurons in each layer ***HP.***: (Epoch, Dropout)

partition algorithm whose objective is set to minimize the communication volume.

Datasets and Models. We evaluate SYLVIE on five real-world large-scale graph benchmarks: Reddit [20], Yelp [16], Ogbn-products [75], Amazon [76], and Ogbn-papers100M [75]. The detailed information is shown in Table V. We choose versatile GNN models commonly adopted in GNN applications for evaluation, including two shallow models GraphSAGE [20] and vanilla GCN [14], deep models GCNII [32], DGANN [31], SGC [33] and JKNet [77], and special GNN model GAT [7] (the number of heads is set to 1). Not that JKNet is only applicable to Reddit dataset and DAGNN is unsuitable to be deployed on Yelp dataset. Regarding the models, we follow the hyperparameter configurations reported in the original papers as closely as possible. The detailed model hyperparameters used for evaluation are presented in Table VI. For JKNet, the number of layers is 8 and hidden size is 128. The training epoch equals to 800 and the dropout rate is 0.5.

Baselines. For the baselines, we compare SYLVIE with four SOTA-distributed full-graph training methods: (1) DGL [37]: the vanilla distributed GNN training on top of the latest DGL 0.9; (2) SAR [40]; (3) PipeGCN [26]; (4) BNS-GCN [22]: the p value is set to 0.1 as suggested by the paper. Baselines are orthogonal to each other in distributed GNN system designs so that we can make a fair comparison. Note that all the baselines do not implement deeper GNNs originally, so we modified deeper GNNs on them ourselves and only show their results on their respective supported GNNs.

Testbeds. Our experiments are performed on two different GPU servers. ❶Severs each with 8 RTX 3090 GPUs (24GB), intra-server connection (CPU-GPU and GPU-GPU) based on PCIe 4.0 lanes and inter-server connection via 1Gbps Ethernet. ❷Servers each with 8 A100 GPUs (80GB) with NVLink and 200Gbps InfiniBand.

A. End-to-end Experiments

We compare the end-to-end performance of SYLVIE with baselines on both RTX 3090 and A100 servers.

Training Speedup and Accuracy Maintenance. Table VII and Figure 9 describe throughput and test accuracy comparisons between SYLVIE and SOTA baselines on versatile GNN models over two 3090 servers. Here throughput is defined as the number of epochs run per second, and we normalize the throughput of each method on base of DGL. In each training task, we treat the first 10 epochs as the warmup stage and only record statistics afterward. We can clearly see that SYLVIE substantially outperforms other methods by a large margin on each dataset and model. Specifically, SYLVIE achieves a marvelous throughput improvement of 8.67∼16.03× over DGL and far exceeds SAR and PipeGCN. We note that PipeGCN does not show significant performance since in the multi-server training, the communication cost is immensely larger than computation and could hardly be hidden.

To further unfold the effectiveness of SYLVIE in distributed setting, we also conduct evaluations on A100 servers with NVLink and 200Gbps InfiniBand, as shown in Table VIII. SYLVIE still shows impressive acceleration and outperforms baselines on such frontier equipment. For the largest dataset Ogbn-papers100M, we partition it to 32 parts and deploy the training on 4 servers (each 8 GPUs). We can see even at such a large-scale setting where communication overhead dominates, SYLVIE still provides the largest speedup and substantially reduces the communication time by 95%.

Generality on Versatile GNNs. Unlike other baselines, SYLVIE consistently performs well in efficiency and model accuracy on deeper and special structured GNNs. In Table VII, SYLVIE always achieves far better training throughput than other methods on all types of GNNs. Especially, SYLVIE successfully converges and maintains model accuracy on deeper and special GNNs, and even reaches higher accuracy in some cases, e.g., enables DAGNN to reach 63.41% on Ogbn-products while achieving the largest throughput 10.06×. On the contrary, current systems fail to accommodate to deeper and special GNNs. For instance, BNS-GCN cannot converge on deeper GNNs at all due to the excessive node dependency loss along layers. Additionally, it incurs a significant accuracy loss of up to 4.9% on GAT, showing its limited generality to other models. PipeGCN also suffers from serious accuracy drop up to 5.45% since the staleness errors accumulate essentially when the model is deep. Via the adaptive optimizations by monitoring training status, SYLVIE is robust to the noise introduced by compressed activations, indicating SYLVIE enables to train deeper and more complex GNNs on large graphs with minimal loss in performance.

Maintaining Model Convergence. We examine the convergence curves of SYLVIE on various models in Figure 10. We can see the curves of SYLVIE are almost identical to that of the original DGL version and converge to high accuracy, verifying SYLVIE preserves model quality well. However, other methods

TABLE VII: Detailed comparison of training throughput and test accuracy between SYLVIE and other baselines when training on two 3090 servers, where the best performance is highlighted in bold. Dash line '-' means the method does not converge. SYLVIE always outperforms others in throughput on all the models and datasets while still achieving high accuracy.

			Reddit		Yelp		Ogbn-products		Amazon	
	Model	**Method**	**Thr.**	**Test Acc.(%)**	**Thr.**	**F1-micro(%)**	**Thr.**	**Test Acc.(%)**	**Thr.**	**Test Acc.(%)**
Shallow	**GraphSAGE**	DGL	1.00×	97.10±0.01	1.00×	65.07±0.19	1.00×	79.19±0.15	1.00×	**81.29**±0.02
		SAR	0.42×	96.02±0.12	0.37×	60.51±0.09	0.64×	74.42±0.07	0.43×	78.85±0.07
		PipeGCN	1.15×	97.02±0.11	1.15×	65.14±0.08	1.19×	**79.29**±0.05	1.05×	81.27±0.08
		BNS-GCN	9.02×	**97.14**±0.01	8.11×	**65.22**±0.23	8.38×	79.11±0.11	9.08×	80.90±0.05
		SYLVIE	**14.64**×	96.87±0.03	**11.27**×	64.92±0.38	**15.74**×	78.85±0.26	**13.70**×	81.24±0.11
	GCN	DGL	1.00×	94.84±0.58	1.00×	47.50±0.07	1.00×	73.70±0.20	1.00×	**56.59**±0.11
		SAR	0.42×	**95.34**±0.17	0.38×	47.00±0.12	0.65×	70.13±0.10	0.43×	53.08±0.07
		PipeGCN	1.15×	94.69±0.56	1.16×	47.16±0.01	1.20×	**74.04**±0.23	1.01×	56.56±0.34
		BNS-GCN	9.18×	95.00±0.33	8.40×	47.27±0.37	8.64×	73.54±0.42	9.34×	56.47±0.60
		SYLVIE	**15.15**×	95.31±0.01	**13.13**×	**47.62**±0.30	**16.03**×	73.78±0.19	**14.61**×	56.07±0.21
Deep	**GCNII**	DGL	1.00×	**89.53**±0.20	1.00×	61.55±0.08	1.00×	**58.34**±0.16	1.00×	42.15±0.21
		PipeGCN	1.14×	84.08±0.32	1.13×	60.18±0.21	1.20×	56.78±0.11	1.03×	41.47±0.18
		BNS-GCN	-	-	-	-	-	-	-	-
		SYLVIE	**17.18**×	89.16±0.11	**12.48**×	**62.43**±0.07	**10.60**×	58.15±0.07	**10.42**×	**43.25**±0.11
	DAGNN	DGL	1.00×	**91.94**±0.20			1.00×	63.22±0.14	1.00×	54.01±0.14
		PipeGCN	-	-	-	-	1.18×	60.32±0.22	1.03×	52.83±0.31
		BNS-GCN	-	-			-	-	-	-
		SYLVIE	**7.88**×	91.89±0.13			**10.06**×	**63.41**±0.12	**12.47**×	**54.91**±0.18
	SGC	DGL	1.00×	80.64±0.19	1.00×	50.30±0.05	1.00×	54.76±0.20	1.00×	**41.12**±0.05
		PipeGCN	1.02×	80.03±0.37	1.12×	49.31±0.12	1.07×	54.08±0.29	1.05×	39.12±0.17
		BNS-GCN	-	-	-	-	-	-	-	-
		SYLVIE	**7.56**×	**80.68**±0.10	**13.46**×	**50.32**±0.07	**12.12**×	**55.02**±0.11	**13.22**×	41.11±0.14
Special	**GAT**	DGL	1.00×	**93.97**±0.60	1.00×	**44.39**±0.16	1.00×	78.14±0.12	1.00×	**42.84**±0.96
		SAR	0.25×	91.47±0.08	0.21×	44.30±0.11	0.27×	76.40±0.06	0.21×	42.48±0.07
		PipeGCN	1.14×	93.85±0.64	1.15×	43.75±0.23	1.19×	77.03±0.11	1.04×	42.37±0.07
		BNS-GCN	7.86×	89.08±0.63	8.11×	43.66±0.24	8.08×	74.07±0.92	8.43×	40.67±0.79
		SYLVIE	**12.26**×	93.40±0.62	**13.48**×	44.15±0.63	**13.21**×	**78.38**±0.18	**8.67**×	42.08±0.25

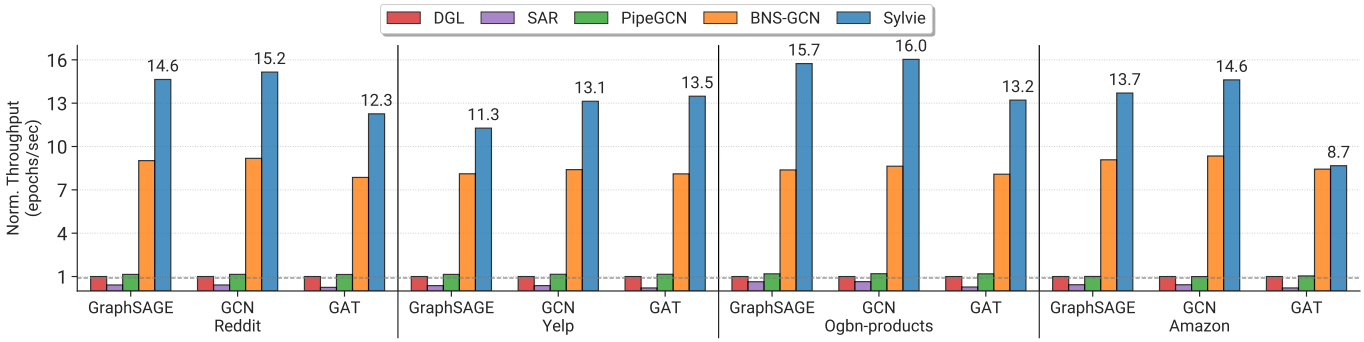

Fig. 9: Training throughput of different methods (normalized to that of DGL, shown in the dashed line) when training three representative models on four datasets on two 3090 servers. SYLVIE outperforms DGL by up to 16.0×.

TABLE VIII: Training epoch time comparison between SYLVIE and other methods on GraphSAGE on A100 servers with NVLink.

Dataset	**Server Setting**	**Method**	**Epoch Time (s)**	**Comm. (s)**
Ogbn-products	2 Servers 16 GPUs	DGL	0.99 (1.00×)	0.87
		PipeGCN	0.73 (1.36×)	0.57
		BNS-GCN	0.39 (2.54×)	0.17
		SYLVIE	**0.23 (4.30×)**	0.11
Ogbn-papers100M	4 Servers 32 GPUs	DGL	17.00 (1.00×)	14.00
		PipeGCN	12.40 (1.37×)	9.70
		BNS-GCN	2.10 (8.10×)	1.47
		SYLVIE	**1.30 (13.08×)**	0.69

TABLE IX: Throughput when training on a single 3090 server.

Model	**Method**	**Dataset**		
		Yelp	**Ogbn-products**	**Amazon**
GraphSAGE (N=8)	DGL	1.00×(1.82 ep./s)	1.00×(0.95 ep./s)	1.00×(0.38 ep./s)
	SAR	0.99×	1.27×	1.12×
	PipeGCN	1.08×	1.05×	0.97×
	BNS-GCN	3.10×	3.07×	6.93×
	SYLVIE	**4.02**×	**4.40**×	**7.78**×
GCN (N=8)	DGL	1.00×(1.91 ep./s)	1.00×(1.12 ep./s)	1.00×(0.42 ep./s)
	SAR	1.04×	1.32×	1.23×
	PipeGCN	1.07×	1.06×	0.94×
	BNS-GCN	2.30×	2.46×	4.78×
	SYLVIE	**4.36**×	**3.44**×	**5.04**×

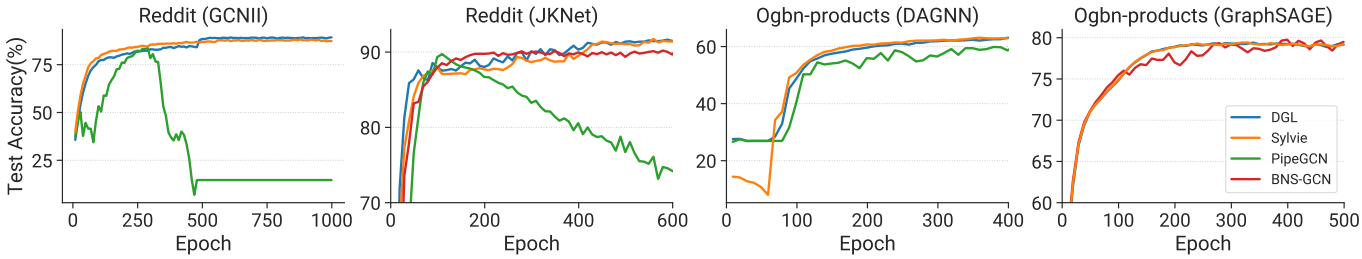

Fig. 10: The convergence curve comparisons of SYLVIE and baselines on different models and datasets over single-server.

TABLE X: Training GraphSAGE on Yelp with different b values and fixed execution on single A100 server.

b	32	8	4	2	1	SYLVIE (adaptive quant)
Epoch Time (s)	0.90	0.52	0.37	0.28	0.22	0.39
Accuracy (%)	65.3	65.3	65.1	64.5	64.4	65.0

TABLE XI: Training GraphSAGE on Yelp with different execution modes and fixed b values on single A100 server.

Method	Fix b=32		Fix b=1	
	Epoch Time (s)	Acc. (%)	Epoch Time (s)	Acc. (%)
Always-sync.	0.90	65.3	0.21	64.4
Always-async.	0.75	64.6	0.12	64.2
SYLVIE (adaptive pipeline)	0.81	64.9	0.17	64.6

either converge to low accuracy (BNS-GCN) or lead to slower convergence and even occur over-fitting (PipeGCN on GCNII and JKNet respectively). The over-fitting is mainly due to PipeGCN's smoothing method, which increases stability on the training set. It constrains the model from exploring a more general minimum point on the test set, thus leading to overfitting on deeper models.

Performance on Single Server. We also test the performance of SYLVIE on a single 3090 server in Table IX. SYLVIE still outperforms other methods in training throughput, with a maximum of 7.78× speedup when training GraphSAGE.

B. Ablation Studies

To verify the effectiveness of our 3D-adaptive scheme and explore the impact of each system module explicitly, we compare SYLVIE with different static settings. All ablation studies are conducted on A100 servers.

Quantization Ablation Study. The evaluations consider different static values of b, from no quantization to 1-bit quantization as shown in Table X. We fix the execution mode to always-synchronous training to make fair comparisons since the adaptive pipeline adjustment is unpredictable in each training. We can see with the decrease of b value, training epoch time also decreases, but with greater accuracy loss. This is because applying low-bit quantization introduces significant variance and degrades accuracy. However, *Quant Orchestrator* enables SYLVIE to gain high throughput (2.3× compared with 32-bit) and maintain robust accuracy (65.0% vs 64.4% of 1-bit). This verifies simply performing static quantization cannot maximize its benefits or keep model quality.

TABLE XII: Epoch communication volume and time breakdown of training GraphSAGE over two servers.

	Method	Comm. Volume(MB)		Per-epoch Time (s)	
		Main Data	Scales	Total	Comm.
Reddit	DGL	2791.7	0	7.28	6.62
	SYLVIE	126.9	15.6	0.5	0.44
Amazon	DGL	5632.6	0	13.33	11.47
	SYLVIE	254.7	30.4	0.97	0.81

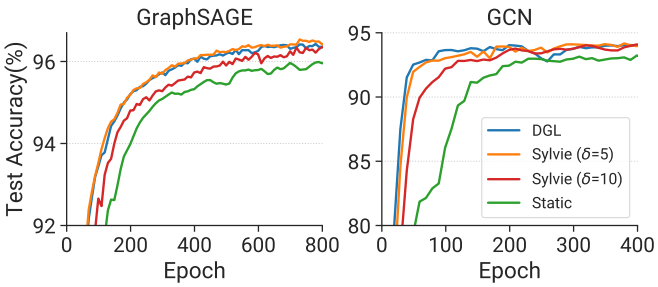

Fig. 11: Sensitivity experiments of comparing range δ on Reddit (N=8, single server).

Pipeline Ablation Study. Here we compare SYLVIE with an always-synchronous and always-asynchronous version. Similarly, we fix b values to make fair comparisons and provide the results in Table XI. We observe that SYLVIE has a larger training speed than always-synchronous version and higher accuracy than always-asynchronous version with comparable speed, which validates *Pipeline Adaptor* successfully strikes efficiency-accuracy trade-off.

Trained on the same model and dataset, the epoch time of 3D-adaptive SYLVIE is 0.27s and accuracy is 65.0%. Together with both ablation studies, we can see SYLVIE combines the best of all worlds from efficiency and accuracy. By dynamically adjusting the system optimizations guided by training status, the 3D-adaptive scheme greatly boosts training while bounding the gradient variance to a limited level, thus reaching a better efficiency-accuracy balance.

C. More Evaluation

Communication Volume and Time. To demonstrate the training speedup is due to the reduced communication, we record the actual communication volume per epoch and training time breakdown in Table XII. We observe that SYLVIE cuts down the communication volume dramatically. For example, there are originally 5632.6 MB of communication per epoch for

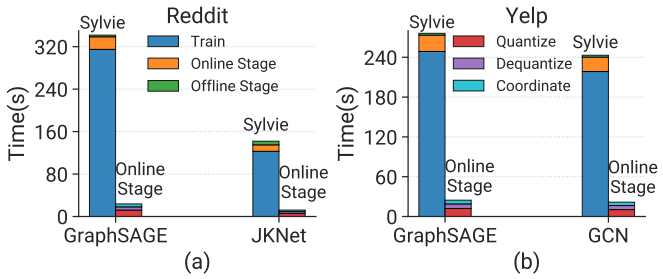

Fig. 12: Wall-clock time of DGL and SYLVIE with overhead.

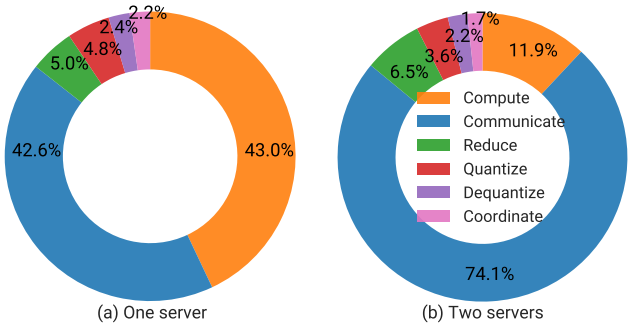

Fig. 13: Ratios of different components in epoch time when training GraphSAGE with SYLVIE on Reddit over single server and two servers. Both quantization and coordination take up negligible overhead.

the Amazon dataset. After deploying SYLVIE, there are only 254.7 MB communicated messages, reducing almost 22× communication volume. Accordingly, the communication time is vastly shortened (from 11.47s to 0.81s).

Sensitivity Analysis. The introduced hyper-parameter δ and λ determine the performance and overhead of SYLVIE. Here we perform sensitivity experiments on δ. As shown in Figure 11, the faster convergence and higher accuracy are obtained when smaller δ value is adopted, but coming with possibly lower throughput (here for GraphSAGE, 4.98 epochs/s with $\delta = 5$ vs. 5.93 epochs/s with $\delta = 10$). Seriously chasing the lowest quantization variance ($\delta = 1$) or just caring about the highest training throughput (very large δ) is not the best choice to fully utilize the benefits of quantization and pipelining. Choosing different values always exists a trade-off between efficiency and accuracy. Currently, we suggest $\lambda = 0.9$ for better convergence, and users can select $\delta = 5$ for higher model quality or larger $\delta = 20$ for faster speed.

System Overhead Analysis. To understand how much extra overhead brought by SYLVIE, we record the time breakdown from two levels: wall-clock time level in Figure 12 and more fine-grained epoch time portions in Figure 13. We can observe both the online and offline overhead are negligible compared with the training time reduction. The wall-clock time of SYLVIE on GraphSAGE-Reddit in Figure 12 is 314.9s, 23.7s and 3.1s for training, online stage and offline stage respectively. The overhead proportion (online and offline stage) in total training time is only 7.8%. Similar conclusions can be obtained from Figure 13. Both cases demonstrate the time consumed by quantization and coordination occupies the smallest portions, indicating the negligible overhead brought by SYLVIE.

TABLE XIII: Epoch time of GraphSAGE on Ogbn-products under different A100 server settings.

Server Setting	Method	Epoch Time (s)	Comm. Time (s)	Comp. Time (s)
1 Server, 8 GPUs	DGL	0.83	0.71	0.09
	SYLVIE	**0.30 (2.8×)**	0.13	0.09
2 Servers, 16 GPUs	DGL	0.99	0.87	0.04
	SYLVIE	**0.23 (4.3×)**	0.11	0.04
3 Servers, 24 GPUs	DGL	1.23	1.13	0.03
	SYLVIE	**0.25 (4.9×)**	0.12	0.03

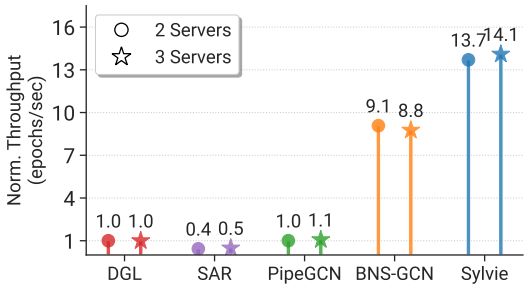

Fig. 14: Normalized training throughput on multiple 3090 servers for GraphSAGE on Amazon.

D. Scalability Analysis on More Servers

To further evaluate SYLVIE's capability, we scale up the training over multiple servers on both server types. Table XIII's results on A100 server show SYLVIE still obtains considerable speedup on high-speed network servers and shows not bad scalability. The speedup rate increases even on more servers, e.g., 2.8∼4.3∼4.9×. Figure 14 presents the normalized training throughput of SYLVIE over 3090 servers. We also observe that SYLVIE maintains great performance and even achieves a higher throughput acceleration ratio when the number of servers increases. On both settings, SYLVIE offers the best training speedup compared with other methods, while SAR and PipeGCN show very limited performance in large-scale training. In a nutshell, SYLVIE can deliver desired performance for larger-scale training scenarios.

VII. CONCLUSION

This work proposes SYLVIE, an efficient distributed GNN training framework that enormously boosts training efficiency in a 3D-adaptive way, while maintaining the model quality. Unlike existing methods which fail to accommodate to universal GNN models, SYLVIE outperforms well on all model structures. Extensive experiments show that SYLVIE can substantially boost the training throughput by up to 17.2×.

ACKNOWLEDGMENTS

We sincerely thank our anonymous ICDE reviewers for their valuable comments on this paper. This research is supported under the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contribution from the industry partner(s).

REFERENCES

[1] F. D. Malliaros and M. Vazirgiannis, "Clustering and community detection in directed networks: A survey," *CoRR*, vol. abs/1308.0971, 2013.

[2] T. Gu, C. Wang, C. Wu, J. Xu, Y. Lou, C. Wang, K. Xu, C. Ye, and Y. Song, "Hybridgnn: Learning hybrid representation in multiplex heterogeneous networks," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 1355–1367.

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

[4] H. Li and L. Chen, "Early: Efficient and reliable graph neural network for dynamic graphs," *Proc. ACM Manag. Data*, vol. 1, p. 1–28, 2023.

[5] J. Chen, J. Gao, and B. Cui, "Ics-gnn+: lightweight interactive community search via graph neural network," *The VLDB Journal*, vol. 32, p. 447–467, 2022.

[6] J. Zeng, P. Wang, L. Lan, J. Zhao, F. Sun, J. Tao, J. Feng, M. Hu, and X. Guan, "Accurate and scalable graph neural networks for billion-scale graphs," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 110–122.

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

[8] N. Yin, F. Feng, Z. Luo, X. Zhang, W. Wang, X. Luo, C. Chen, and X.-S. Hua, "Dynamic hypergraph convolutional network," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 1621–1634.

[9] X. Qin, N. Sheikh, C. Lei, B. Reinwald, and G. Domeniconi, "Seign: A simple and efficient graph neural network for large dynamic graphs," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 2850–2863.

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

[11] G. Lv and L. Chen, "On data-aware global explainability of graph neural networks," *Proceedings of the VLDB Endowment*, vol. 16, p. 3447–3460, 2023.

[12] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '18, 2018.

[13] Z. Liu, K. Zhou, F. Yang, L. Li, R. Chen, and X. Hu, "Exact: Scalable graph neural networks training via extreme activation compression," in *International Conference on Learning Representations*, ser. ICLR '21, 2021.

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

[15] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, 2019.

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

[17] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, ser. NIPS'19, 2019.

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

[19] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS'18, 2018.

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

[21] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of graph neural networks with roc," in *Proceedings of Machine Learning and Systems*, ser. MLSys '20, 2020.

[22] C. Wan, Y. Li, A. Li, N. S. Kim, and Y. Lin, "Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling," in *Proceedings of Machine Learning and Systems*, ser. MLSys '22, 2022.

[23] J. Thorpe, Y. Qiao, J. Eyolfson, S. Teng, G. Hu, Z. Jia, J. Wei, K. Vora, R. Netravali, M. Kim, and G. H. Xu, "Dorylus: Affordable, scalable, and accurate GNN training with distributed CPU servers and serverless threads," in *15th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI '21, 2021.

[24] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, "Distdgl: Distributed graph neural network training for billion-scale graphs," *CoRR*, vol. abs/2010.05337, 2021.

[25] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: a comprehensive graph neural network platform," *Proceedings of the VLDB Endowment*, vol. 12, pp. 2094–2105, 2019.

[26] C. Wan, Y. Li, C. R. Wolfe, A. Kyrillidis, N. S. Kim, and Y. Lin, "Pipegcn: Efficient full-graph training of graph convolutional networks with pipelined feature communication," in *International Conference on Learning Representations*, ser. ICLR '22, 2022.

[27] J. Peng, Z. Chen, Y. Shao, Y. Shen, L. Chen, and J. Cao, "Sancus: staleness-aware communication-avoiding full-graph decentralized training in large-scale graph neural networks," *Proceedings of the VLDB Endowment*, vol. 15, p. 1937–1950, 2022.

[28] S. Yang, M. Zhang, W. Dong, and D. Li, "Betty: Enabling large-scale gnn training with batch-level graph partitioning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. Association for Computing Machinery, 202023.

[29] Z. Lin, C. Li, Y. Miao, Y. Liu, and Y. Xu, "Pagraph: Scaling gnn training on large graphs via computation-aware caching," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC '20. Association for Computing Machinery, 2020.

[30] C. Zheng, H. Chen, Y. Cheng, Z. Song, Y. Wu, C. Li, J. Cheng, H. Yang, and S. Zhang, "Bytegnn: efficient graph neural network training at large scale," *Proceedings of the VLDB Endowment*, vol. 15, p. 1228–1242, 2022.

[31] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '20. Association for Computing Machinery, 2020.

[32] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML '20, 2020, pp. 1725–1735.

[33] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. ICML '19, 2019, pp. 6861–6871.

[34] M. Zhang, Q. Hu, P. Sun, Y. Wen, and T. Zhang, "Boosting distributed full-graph gnn training with asynchronous one-bit communication," *CoRR*, vol. abs/2303.01277, 2023.

[35] Q. Wang, Y. Zhang, H. Wang, C. Chen, X. Zhang, and G. Yu, "Neutronstar: Distributed gnn training with hybrid dependency management," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22. Association for Computing Machinery, 2022.

[36] G. Li, C. Xiong, A. Thabet, and B. Ghanem, "Deepergcn: All you need to train deeper gcns," *CoRR*, vol. abs/2006.07739, 2020.

[37] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *CoRR*, vol. abs/1909.01315, 2020.

[38] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *CoRR*, vol. abs/1903.02428, 2019.

[39] A. Tripathy, K. Yelick, and A. Buluç, "Reducing communication in graph neural network training," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

[40] H. Mostafa, "Sequential aggregation and rematerialization: Distributed full-batch training of graph neural networks on large graphs," in *Proceedings of Machine Learning and Systems*, ser. MLSys '22, 2022.

[41] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, "NeuGraph: Parallel deep neural network computation on large graphs," in *2019 USENIX Annual Technical Conference*, ser. USENIX ATC '19, 2019.

[42] Y. Wang, B. Feng, G. Li, S. Li, L. Deng, Y. Xie, and Y. Ding, "GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration

on GPUs," in *15th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI '21. USENIX Association, 2021, pp. 515–531.

[43] B. Wan, J. Zhao, and W. Chuan, "Adaptive message quantization and parallelization for distributed full-graph gnn training," in *Proceedings of Machine Learning and Systems*, ser. MLSys '23, 2023.

[44] Y. Zhang and A. Kumar, "Lotan: Bridging the gap between gnns and scalable graph analytics engines," *Proceedings of the VLDB Endowment*, vol. 16, p. 2728–2741, 2023.

[45] X. Wan, K. Xu, X. Liao, Y. Jin, K. Chen, and X. Jin, "Scalable and efficient full-graph gnn training for large graphs," *Proc. ACM Manag. Data*, vol. 1, p. 1–23, 2023.

[46] G. V. Demirci, A. Haldar, and H. Ferhatosmanoglu, "Scalable graph convolutional network training on distributed-memory systems," *Proceedings of the VLDB Endowment*, vol. 16, p. 711–724, 2022.

[47] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017.

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

[49] X. Miao, Y. Wang, Y. Jiang, C. Shi, X. Nie, H. Zhang, and B. Cui, "Galvatron: Efficient transformer training over multiple gpus using automatic parallelism," *Proceedings of the VLDB Endowment*, vol. 16, p. 470–479, 2022.

[50] X. Miao, Y. Shi, Z. Yang, B. Cui, and Z. Jia, "Sdpipe: A semi-decentralized framework for heterogeneity-aware pipeline-parallel training," *Proceedings of the VLDB Endowment*, vol. 16, p. 2354–2363, 2023.

[51] S. Zhang, A. Sohrabizadeh, C. Wan, Z. Huang, Z. Hu, Y. Wang, J. Cong, Y. Sun *et al.*, "A survey on graph neural network acceleration: Algorithms, systems, and customized hardware," *arXiv preprint arXiv:2306.14052*, 2023.

[52] M. Ramezani, W. Cong, M. Mahdavi, M. Kandemir, and A. Sivasubramaniam, "Learn locally, correct globally: A distributed algorithm for training graph neural networks," in *International Conference on Learning Representations*, ser. ICLR '22, 2022.

[53] Z. Song, Y. Gu, J. Qi, Z. Wang, and G. Yu, "Ec-graph: A distributed graph neural network system with error-compensated compression," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 648–660.

[54] Y. Ma, P. Gong, J. Yi, Z. Yao, C. Li, Y. He, and F. Yan, "Bifeat: Supercharge gnn training via graph feature quantization," *CoRR*, vol. abs/2207.14696, 2023.

[55] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," *CoRR*, vol. abs/2008.05000, 2021.

[56] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018.

[57] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '18, 2018.

[58] J. Chen, L. Zheng, Z. Yao, D. Wang, I. Stoica, M. Mahoney, and J. Gonzalez, "Actnn: Reducing training memory footprint via 2-bit activation compressed training," in *Proceedings of the 38th International Conference on Machine Learning*, ser. ICML '21, 2021.

[59] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '17, 2017.

[60] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '17, 2017.

[61] Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing, "Pipe-sgd: A decentralized pipelined sgd framework for distributed deep net training," 2018.

[62] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '14, 2014.

[63] B. Feng, Y. Wang, X. Li, S. Yang, X. Peng, and Y. Ding, "Sgquant: Squeezing the last bit on graph neural networks with specialized quantization," in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2020.

[64] Y. Zhao, D. Wang, D. Bates, R. Mullins, M. Jamnik, and P. Lio, "Learned low precision graph neural networks," *CoRR*, vol. abs/2009.09232, 2020.

[65] Y. Wang, B. Feng, and Y. Ding, "Qgtc: accelerating quantized graph neural networks via gpu tensor core," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '22, 2022.

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

[67] F. Zhu, R. Gong, F. Yu, X. Liu, Y. Wang, Z. Li, X. Yang, and J. Yan, "Towards unified int8 training for convolutional neural network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, ser. CVPR '20, June 2020.

[68] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, ser. NeurIPS '15, 2015.

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

[70] D. Jhunjhunwala, A. Gadhikar, G. Joshi, and Y. C. Eldar, "Adaptive quantization of model updates for communication-efficient federated learning," in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 3110–3114.

[71] R. Hönig, Y. Zhao, and R. Mullins, "DAdaQuant: Doubly-adaptive quantization for communication-efficient federated learning," in *Proceedings of the 39th International Conference on Machine Learning*, ser. ICML '22, 2022, pp. 8852–8866.

[72] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proceedings of the 35th International Conference on Machine Learning*, ser. ICML '18, 2018, pp. 3043–3052.

[73] P. Stock, A. Fan, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin, "Training with quantization noise for extreme model compression," in *International Conference on Learning Representations*, ser. ICLR '20, 2020.

[74] Y. Dong, R. Ni, J. Li, Y. Chen, J. Zhu, and H. Su, "Learning accurate low-bit deep neural networks with stochastic quantization," *CoRR*, vol. abs/1708.01001, 2017.

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

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

[77] K. Xu, C. Li, Y. Tian, T. Sonobe, K. ichi Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks," *CoRR*, vol. abs/1806.03536, 2018.