Expediting Distributed GNN Training with Feature-only Partition and Optimized Communication Planning

Bingqian Du*, Jun Liu*, Ziyue Luo†, Chuan Wu‡, Qiankun Zhang§, and Hai Jin*
* National Engineering Research Center for Big Data Technology and System,
Services Computing Technology and System Lab, Cluster and Grid Computing Lab,
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
† Department of Electrical and Computer Engineering, The Ohio State University, USA
‡ Department of Computer Science, The University of Hong Kong, Hong Kong
§ School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China
E-mail: *{bqdu, liujun2023, hjin}@hust.edu.cn, †luo.1457@osu.edu, ‡cwu@cs.hku.hk, §qiankun@hust.edu.cn

***Abstract*—Feature-only partition of large graph data in distributed *Graph Neural Network* (GNN) training offers advantages over commonly adopted graph structure partition, such as minimal graph preprocessing cost and elimination of cross-worker subgraph sampling burdens. Nonetheless, performance bottleneck of GNN training with feature-only partitions still largely lies in the substantial communication overhead due to cross-worker feature fetching. To reduce the communication overhead and expedite distributed training, we first investigate and answer two key questions on convergence behaviors of GNN model in feature-partition based distribute GNN training: 1) As no worker holds a complete copy of each feature, can gradient exchange among workers compensate for the information loss due to incomplete local features? 2) If the answer to the first question is negative, is feature fetching in every training iteration of the GNN model necessary to ensure model convergence? Based on our theoretical findings on these questions, we derive an optimal communication plan that decides the frequency for feature fetching during the training process, taking into account bandwidth levels among workers and striking a balance between model loss and training time. Extensive evaluation demonstrates consistent results with our theoretical analysis, and the effectiveness of our proposed design.**

I. Introduction

Graph neural networks (GNN) [1] have been proposed in recent years that operate on graph structured data, generate low-dimensional embeddings for nodes by aggregating information from neighborhoods, and have catalyzed breakthroughs in many graph-related tasks (e.g., node classification [2], link prediction [3] [4], and graph classification [5]). The success of GNNs stems from efficiently exploiting the following graph information [6]: (i) graph structure, which describes interdependencies among nodes and gives the L-hop neighborhoods involved in embedding calculation of a single node; (ii) high-dimensional node feature, which is an extensive array of attributes that provides detailed descriptions of each node and serves as input for node embedding calculation. For example, a 602-dimensional feature is used for each node (post), when training GraphSage model [2] on the Reddit posts dataset for topical community prediction.

This work was supported by NSFC under grants 62302187, 62302183 and by Hong Kong RGC grants 17207621, 17203522 and C7004-22G (CRF). (Corresponding author: Qiankun Zhang.)

Real-world graph datasets are large, often including millions to billions of nodes and edges. For example, the user-to-item graph collected by Pinterest consists of 3 billion nodes and 18 billion edges, with an approximate size of 18TB [7]. Storing and computing such a large graph may well exceed the memory and computation capacities of a single device or physical machine (i.e., a worker). Distributed GNN training has thus been adopted [8] [9], which partitions a large graph among multiple workers, trains embeddings of a subset of the graph nodes on each worker, and synchronizes model gradients among GNN model replicas on the workers. The current distributed GNN training systems commonly adopt **structure partition** of the large graph [9] [10], that partitions the graph structure among workers and stores features of nodes in each partition on the respective worker. The goal is to balance worker workload and minimize cross-worker edges. This graph partition scheme ensures that each node, along with its feature vector, is assigned to at least one worker.

Upon a thorough analysis of this graph partition paradigm, we have made three key observations: (1) Graph structure partitioning consumes considerable time especially on large graphs. For example, it takes the widely adopted balanced min-edge-cut partitioning method, METIS [11], about an hour to partition the ogbn-papers100M dataset [12]. (2) Cross-worker subgraph sampling combined with feature fetching significantly slows down the training process and results in low GPU utilization [13]. The feature communication among workers can be unbalanced too, when one worker hosts a large number of nodes requested by other workers in a training iteration, creating throughput bottlenecks. (3) The aforementioned issue is further exacerbated with the increase of GNN layers. As the number of GNN layers increases, the number of nodes involved in sampled subgraphs increases exponentially, leading to significantly larger communication overhead.

A real-world graph dataset typically consists of the adjacency matrix denoting the graph structure and a high-dimensional feature matrix of all nodes. The size of the graph structure is usually much smaller than that of the features, e.g., the structure information of ogbn-papers100M occupies 24GB, while its feature matrix consumes 53GB memory [12]. Therefore, the graph structure is more likely to fit within the memory of a physical machine [14].

Given the drawbacks of structure partition and the fact that graph structure can usually fit within the memory of a machine, an alternative graph data partition scheme for distributed GNN training has been advocated [14]: instead of partitioning the graph structure and storing complete features of nodes with the respective partitions, the feature vector of each node is partitioned and dispersed evenly among all workers, i.e., each worker holds $\frac{d}{K}$ dimensions of feature vectors of all nodes where d is the total dimension number of each node's feature vector and K is the number of workers, while the complete graph structure is stored in each physical machine. This paradigm sidesteps the hassle of graph structure partitioning, avoids the communication overhead for cross-machine subgraph sampling, and can achieve strictly balanced communication among workers for feature fetching.

Nevertheless, the average feature transfer size among workers in this **feature-partition** based distributed GNN training remains similar compared to structure-partition based distributed GNN training. Therefore, feature communication still renders the major bottleneck during the training process. A natural question is: how can we further reduce the communication overhead due to feature fetching in feature-partition based distributed GNN training?

The main challenge to answer this question is that both the convergence behavior of the GNN model and its relation with feature communication have not been explored under feature-partition based paradigm. In this paper, we investigate the effect of feature communication on the convergence performance of GNN models in feature-partition based distributed GNN training and propose an optimal feature communication plan based on our findings, to hit the sweet spot between model convergence error and training time. The contributions we make in this paper can be summarized as follows.

▷ We analyze the convergence behavior of the GNN model in the extreme case of "gradient only communication", where only gradients calculated on incomplete local features are exchanged among workers during the feature-partition based distributed GNN training, and corroborate that the convergence bound of the GNN model is subject to a non-vanishing term closely related to the incomplete feature information at each worker. This result indicates that *the communication of feature dimensions scattered across all workers is indispensable*; otherwise, the GNN model fails to converge.

▷ To guarantee the desired convergence, we enable each worker to periodically invoke feature fetching from all other workers for constructing complete feature vectors of nodes in its sampled subgraphs. Through establishing a relationship between convergence error and training completion time, we ascertain that the model's convergence error follows a convex function with respect to the feature communication frequency. Based on the insights obtained, we derive an optimal feature communication plan, which achieves the best tradeoff between model performance and training completion time.

▷ We thoroughly evaluate our theoretical analysis and optimal feature communication plan via extensive experiments, by training representative GNN models on various real-world large graph datasets under different settings. The results are consistent with our theoretical implications and demonstrate that feature-partition based distributed GNN training with our optimal feature communication plan can achieve an average speedup of 1.5× for model convergence to the same accuracy.

II. Motivations and Related Works

A. Graph Neural Networks

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Each vertex $v \in \mathcal{V}$ has a feature vector $x_v \in \mathbb{R}^d$, where d is the feature dimension. The feature vectors of all nodes constitute the feature matrix $X \in \mathbb{R}^{\mathcal{N} \times d}$, where $\mathcal{N}$ is the number of vertices. We use $\mathcal{N}_1(v)$ to represent the set of one-hop neighbor nodes of node v. A GNN calculates the embedding of a node in a recursive manner [15]:

$$h_v^{(l)} = \text{Update}(h_v^{(l-1)}, \text{Aggregate}(h_u^{(l-1)}, \forall u \in \mathcal{N}_1(v))) \quad (1)$$

where $h_v^{(l)}$ is the representation (aka embedding) of node v after computing by l layers of the GNN, and $h_v^{(0)} = x_v, \forall v \in \mathcal{V}$. GNNs differ in their choices of Update$(\cdot)$ and Aggregate$(\cdot)$ in (1) [16]. For example, GraphSage [2] uses mean, LSTM or pooling for Aggregate$(\cdot)$ and a linear layer for Update$(\cdot)$.

In each training iteration, subgraphs enclosing sampled L-hop neighbors of the training nodes (i.e., nodes whose embeddings are to be computed) are constructed, each serving as a sample in the training. The sampled subgraphs and features of nodes in the subgraphs are fed as input to the GNN model.

B. Distributed GNN Training

In distributed GNN training, the graph data is partitioned among multiple devices or physical machines. Each worker (usually on one device such as a GPU) hosts a copy of the GNN model and processes a subset of the training nodes in each training iteration, i.e., a mini-batch, by sampling L-hop subgraphs, fetching node features, and learning embeddings.

1) Structure-partition based Distributed GNN Training: The current distributed GNN training frameworks such as DGL [17] and BGL [13] mostly adopt **structure partition** of the graph data. The workflow of these system goes as follows: (1) *graph partition*, where the graph structure and features are partitioned (Fig. 1(a)), with every node, along with its feature vector, allocated to at least one worker.[1] The training proceeds through many iterations, each with three stages at each worker: (2.1) *subgraph sampling*, where every worker communicates with others to construct L-hop neighborhood subgraphs of training nodes in its mini-batch, which may well reside on other workers; (2.2) *feature fetching*, where feature vectors of remotely stored nodes in sampled subgraphs are retrieved from other workers; (2.3) *model computation*, where forward and backward computation is performed on the local GNN model,

[1]We obfuscate the implementation intricacies, deliberately omitting the mention of components such as graph store server and sampler, to accentuate the overarching concept at a higher level.

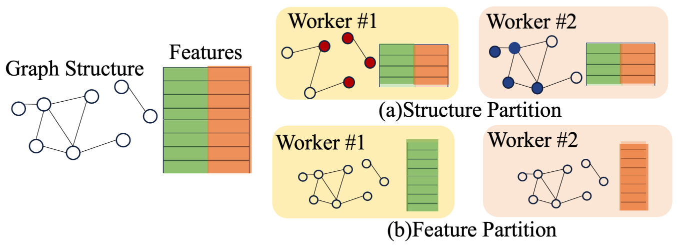


Fig. 1: Distributed GNN Training Paradigms

and gradients of the GNN model are synchronized among the workers for global model update. We identify two key issues in structure-partition based distributed GNN training that have not been fully addressed or been overlooked by existing works.

Problem 1: Extra cost due to graph partitioning. We evaluate three widely adopted graph partitioning algorithms in existing distributed GNN/graph processing systems: (a) random partitioning supported by Euler [18] and DGL [17]; (b) METIS [11], the default partitioning strategy in DGL, which is a balanced min-edge-cut method; (c) *Block-based Deterministic Greedy* (BDG) partitioning algorithm proposed by G-Miner [19], where multiple *breadth-first searches* (BFS) are performed starting from randomly selected source nodes. Each node traversed by a BFS is then assigned to a specific block/partition. Table I gives the partitioning time (T_{par}), peak memory consumption and job completion time (training time required to achieve a model test accuracy greater than 76%, under the condition that the accuracy variance over 30 consecutive training iterations is smaller than 0.0001) when training a GraphSage model on ogbn-products dataset using DGL [2] across 4 workers with our proposed method under a 10Gbps inter-connect bandwidth. The partition process imposes significant computation overhead and memory consumption, and its corresponding runtime may well exceed the total training time. This inspires us to reconsider the necessity of graph structure partition in distributed GNN training.

TABLE I: Performance of Graph Partition Methods

Dataset	Method	T$_{par}$(s)	Memory(GB)	JCT(s)
ogbn-products	Random	92.84	36.04	11.2
	METIS	106.34	32.33	
	BDG	57.99	25.16	

In addition, quite a few partitioning algorithms, including RandomVertexCut [20] and GRID [21], fail to scale to large graphs [13] [14] of similar sizes as ogbn-papers100M, due to high memory consumption. The objective of graph partitioning is typically to minimize cross-worker edges. However, the reduction of cross-worker communication offered by graph partitioning diminishes quickly as the number of GNN layers increases. Recall that a L-layer GNN computes a node's embedding recursively using information of L-hop neighbor nodes. Nodes needed in sampled subgraphs may well cover the complete graph as the number of GNN layer increases, and any graph partitioning algorithm would fall short in reducing the heavy cross-worker communication overhead then.

Problem 2: Performance bottlenecks owing to cross-worker subgraph sampling and node feature transmission. Fig. 2 shows the training time breakdown of one mini-batch with different batch sizes, when training the GraphSage model with structure partitioning of ogbn-products dataset [12] across 4 workers. More than 80% of the training time is spent on cross-worker subgraph sampling and feature fetching. This excessive communication overhead originates primarily from inter-dependencies among nodes in algorithmic design (Eqn. (1)) of GNN. Moreover, if a substantial portion of nodes in sampled subgraphs reside on a single worker, the worker can become a communication bottleneck, further prolonging the feature fetching time. In Fig. 3, the average GPU utilization is computed as the mean value of real-time GPU utilization over several 5-second intervals; the approximate 20% GPU utilization aligns with the time breakdown findings.

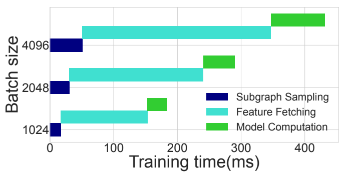


Fig. 2: Training Time Breakdown

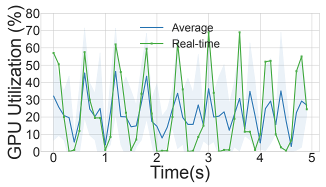


Fig. 3: GPU Utilization

Reducing the cost associated with cross-worker subgraph sampling and feature transmission is the key to improve the efficiency of distributed GNN training.

2) Feature-partition based Distributed GNN Training: In feature-partition based distributed GNN training (Fig. 1(b)), **feature partition** is conducted, with the dimensions of feature vectors evenly partitioned and distributed among all workers. That is, each worker holds $\frac{d}{K}$ dimensions of the feature vectors of all nodes. The complete graph structure is stored within each physical machine, accessible by workers on the same machine. The training process is similar to structure-partition based distributed GNN training, except that the L-hop subgraphs can be solely sampled at each worker, and missing dimensions of the needed node features are fetched from other workers in each training iteration.

This paradigm exhibits superiority in the following three aspects: (1) little extra overhead for graph preprocessing. This feature partition is performed on regular vectors instead of the irregular graph structure, very easy to achieve and requiring minimal extra time and memory space; (2) evenly distributed communication in each training iteration. Given that the feature dimensions are evenly distributed among workers, the size of feature data transmitted from each of the other workers to a specific worker is the same, i.e., $n_k\frac{d}{K}$, where n_k is the number of nodes in the subgraphs sampled at this worker; (3) eliminated overhead for cross-worker subgraph sampling, since the graph structure is stored locally for each worker.

On the other hand, the substantial overhead of cross-worker feature transmission still exits in feature-partition based distributed GNN training. The average size of cross-worker feature transfers in each training iteration is the same between feature-partition based and structure-partition based paradigms, if the nodes are uniformly distributed among workers in structure-partition based distributed GNN training. Suppose the total number of nodes involved in sampled subgraphs of all workers in a training iteration is N. Then the total feature transfer size is $\frac{(K-1)Nd}{K}$ for both paradigms.

This naturally leads to the key question that we are addressing in this paper: ***Can we further minimize the overhead of cross-worker feature fetching in feature-partition based distributed GNN training paradigm?***

Providing a definitive answer to this question is not straightforward. While feature caching and neighborhood sampling (sample only a subset of L-hop neighbor nodes) can alleviate this overhead, their benefits are constrained by the cache size and the number of GNN layers. We aim to propose a novel approach that can independently reduce the feature fetching overhead effectively while synergizing seamlessly with caching and neighbor sampling. The approximate 80% of GPU waiting time in Fig. 3 suggests that, *if feature fetching is carried out periodically, e.g., every τ iterations with $\tau \geq 4$, then the communication time could potentially be fully overlapped with computation time*. Inspired, we explore the feasibility of omitting some feature fetching steps in feature-partition based distributed GNN training.

III. Impact of Feature Fetching

We first identify the impact of remote feature fetching on the performance of the trained GNN model. We consider a distributed GNN training job compromising K workers. For presentation simplicity, we assume each worker is running on a physical machine with a GPU for model computation and host memory to store the graph structure and feature partitions. Recall that the graph dataset $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of $\mathcal{N}$ nodes, and provides a feature matrix $X \in \mathbb{R}^{\mathcal{N} \times d}$, with each row being the feature vector of node v, $x_v \in \mathbb{R}^d$.

The feature-partition paradigm is employed in the distributed GNN training job, with the feature matrix X evenly partitioned column-wise and distributed among the K workers. As a result, each worker $k \in [K]$ ($[K] = \{1, 2, ..., K\}$) locally retains a feature matrix $X^k \in \mathbb{R}^{\mathcal{N} \times \frac{d}{K}}$, that includes columns $[\frac{d}{K} * (k-1) : \frac{d}{K} * k)$ in the complete feature matrix X. Each row of X^k, denoted as $x_v^k \in \mathbb{R}^{\frac{d}{K}}$, is the local feature vector of node $v, \forall v \in \mathcal{V}$. The goal of the distributed GNN training job is to collectively learn a GNN model, minimizing the following loss function, where w denotes GNN model parameters and $h_v^{(L)}$ is calculated according to (1).

$$\min_w \mathcal{L}(w) = \frac{1}{\mathcal{N}} \sum_{v \in \mathcal{V}} \mathcal{L}(h_v^{(L)}, w) \quad (2)$$

Commonly in each training iteration t of distributed GNN training, every worker fetches features from others to reconstruct the complete feature vectors for nodes involved in its sampled subgraphs, and the GNN model parameters are updated in the following unbiased manner:

$$w_{t+1} = w_t - \eta g_t = w_t - \eta \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|B_t^k|} \sum_{v \in B_t^k} \nabla \mathcal{L}(h_v^{(L)}, w_t) \quad (3)$$

where g_t represents the average gradients of model parameters among all workers computed on complete feature vectors, η is the learning rate, and B_t^k represents the mini-batch of training nodes at worker k in training iteration t.

We examine the feasibility of reducing feature fetching frequency during iterative GNN training, through a comprehensive understanding of how feature fetching affects the convergence behavior of the GNN model in feature-partition based distributed training. We investigate two questions:

(1) Considering that model gradients containing local feature information are exchanged among workers in each training iteration for global GNN model update, is feature fetching truly indispensable?

(2) If feature fetching is indispensable, is it feasible not to conduct it in every training iteration?

A. The Case without Cross-worker Feature Fetching

To answer the first question, we examine the extreme case that each worker k trains the GNN model and computes the model gradients solely based on its local feature matrix X^k throughout the training process, with gradients being the only exchanged information among workers. Model parameter update in this case is:

$$w_{t+1} = w_t - \eta \tilde{g}_t = w_t - \eta \frac{1}{K} \sum_{k \in [K]} \frac{1}{|B_t^k|} \sum_{v \in B_t^k} \nabla \mathcal{L}(\tilde{h}_v^{(L)}(k), w_t) \quad (4)$$

where $\tilde{g}_t$ denotes the average gradients of model parameters across all workers computed based on locally retained feature vectors, and $\tilde{h}_v^{(L)}(k)$ is calculated following (1) using feature vector $\tilde{x}_u^k \in \mathbb{R}^d, \forall u \in \mathcal{V}$. $\tilde{x}_u^k$ is a d-dimensional vector with locally maintained feature dimensions of worker k filled into their corresponding positions as in the complete feature vector x_u and all other dimensions filled with 0. Specifically, $\tilde{x}_u^k[i] = x_u^k[i - \frac{d}{K} \times (k-1)]$, if $i \in [\frac{d}{K} \times (k-1), \frac{d}{K} \times k)$, and 0, otherwise. $\tilde{X}^k$ represents the feature matrix composed by $\tilde{x}_u^k$, $\forall u \in \mathcal{V}$. We only make the basic assumption used in standard analysis of *stochastic gradient descent* (SGD) algorithm [22] [23], to derive convergence of the GNN model[2].

Assumption 1 (Smoothness). *The loss objective function is Lipschitz smooth with respect to model parameters, that is, $\mathcal{L}(y) \leq \mathcal{L}(y') + \langle \nabla \mathcal{L}(y'), y - y' \rangle + \frac{L_f}{2} ||y - y'||^2, \forall y, y' \in$ dom $\mathcal{L}$, where L_f is the Lipschitz constant.*

We derive the following theorem on model convergence of GNN training without cross-worker feature fetching.

Theorem 1 (Convergence with Non-vanishing Errors). *Under Assumption 1, the minimum average gradient norm of GNN training that follows the update rule in (4) is bounded by a* ***non-vanishing bias term****, even with sufficiently many training iterations and a learning rate close to zero. Specifically, if the learning rate $\eta \leq \frac{1}{L_f}$, then optimization error is bounded as:*

$$\min_{t \in [T]} E[||\nabla \mathcal{L}(w_t)||^2] \leq \frac{2}{\sqrt{T}} (\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \frac{L_f}{\sqrt{T}} \sigma_{var} + \sigma_{bias} \quad (5)$$

where T is the number of training iterations, w_1 denotes the initial model parameters and w^* represents the optimal model parameters minimizing loss function (2). $\sigma_{var} = \max_{t \in [T]} E[||\tilde{g}_t - E[\tilde{g}_t]||^2]$ is the gradient variance caused by mini-batch sampling and equal to 0 if $B_t^k = \mathcal{V}$, and $\sigma_{bias} = \max_{t \in [T]} E[||\nabla \mathcal{L}(w_t) - E[\tilde{g}_t]||^2]$ is the bias representing the difference between the gradients computed on $h_v^{(L)}$ and $\tilde{h}_v^{(L)}(k)$ $\forall k \in [K]$, respectively.

[2]All the theoretical results presented below do not depend on the specific method used to partition features among workers.

Proof. We provide a proof sketch here and leave the full proof in appendix. Conditioned on w_t and $\eta \leq \frac{1}{L_f}$, we could derive the following relationship based on assumption 1 by letting $\sigma_{bias}^t = \nabla\mathcal{L}(w_t) - E[\tilde{g}_t]$ and $\sigma_{var}^t = \tilde{g}_t - E[\tilde{g}_t]$.

$$E[\mathcal{L}(w_{t+1})] \leq \mathcal{L}(w_t) + \langle \nabla\mathcal{L}(w_t), E[w_{t+1} - w_t]\rangle + \frac{L_f}{2}E[||w_{t+1} - w_t||_F^2]$$
$$\overset{(a):\eta \leq \frac{1}{L_f}}{\leq} \mathcal{L}(w_t) + \frac{\eta}{2}(-2\langle \nabla\mathcal{L}(w_t), \nabla\mathcal{L}(w_t) - \sigma_{bias}^t\rangle$$
$$+ E[||\nabla\mathcal{L}(w_t) - \sigma_{bias}^t||_F^2]) + \frac{\eta^2 L_f}{2}E[||\sigma_{var}^t||_F^2] \quad (6)$$

Summing up (6) from $t = 1$ to T and taking expectation with respect to the randomness in mini-batch sampling,

$$\frac{1}{T}\sum_{t=1}^{T} E[||\nabla\mathcal{L}(w_t)||_F^2] \leq \frac{2}{\eta T}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \sigma_{bias} + \eta L_f \sigma_{var}$$
$$\overset{(b)}{=} \frac{2}{\sqrt{T}}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \sigma_{bias} + \frac{L_f}{\sqrt{T}}\sigma_{var} \quad (7)$$

where (b) holds when T is sufficiently large and $\eta = \frac{1}{\sqrt{T}}$. □

Indispensable Feature Fetching. Theorem 1 reveals that without cross-worker feature fetching, the GNN training cannot be guaranteed to converge to a stationary point, i.e., the local optimum,[3] of the objective function (2), due to the bias error σ_{bias} that does not decrease over model update iterations. This implies that gradient exchange among workers cannot compensate for the information loss of incomplete local features. Feature fetching is indispensable to ensure model training convergence to optimality.

We further establish an upper bound of σ_{bias} to clarify the direct relationship between this irreducible error and incomplete local features. σ_{bias} also acts as a lower bound of the gradient norm and hence genuinely affects model convergence.

1) Upper Bound of σ_{bias}: Here we restrict our discussion to the representative GNN model, *graph convolutional networks* (GCNs) [24] of the following form

$$Z^{(l)} = PH^{(l-1)}w^{(l)} \quad H^{(l)} = \sigma(Z^{(l)}) \quad (8)$$

where $H^{(l)}$ is the layer-l representation matrix, each row of which, $h_v^{(l)}$, is the layer-l embedding of a graph node v. P is the propagation matrix defining the inter-dependencies among nodes, which is a normalized version of the adjacency matrix, and $\sigma(\cdot)$ is the activation function. We use some mild assumptions to establish a connection between the gradients and the feature matrix X on which the gradients are computed.

Assumption 2 (Lipschitz Continuity of Loss and Activation)**.** *There exist $L_l > 0$ and $S_l > 0$ such that*

$$\begin{aligned} ||\mathcal{L}(h_v^{(L)}, w) - \mathcal{L}(\tilde{h}_v^{(L)}, w)||^2 &\leq L_l||h_v^{(L)} - \tilde{h}_v^{(L)}||^2 \\ ||\nabla_{h_v^{(L)}}\mathcal{L}(h_v^{(L)}, w) - \nabla_{\tilde{h}_v^{(L)}}\mathcal{L}(\tilde{h}_v^{(L)}, w)||^2 &\leq S_l||h_v^{(L)} - \tilde{h}_v^{(L)}||^2 \end{aligned} \quad (9)$$

There exist $L_c > 0$ and $S_c > 0$ such that

$$\begin{aligned} ||\sigma(z_v^{(l)}) - \sigma(\tilde{z}_v^{(l)})||^2 &\leq L_c||z_v^{(l)} - \tilde{z}_v^{(l)}||^2 \\ ||\dot{\sigma}(z_v^{(l)}) - \dot{\sigma}(\tilde{z}_v^{(l)})||^2 &\leq S_c||z_v^{(l)} - \tilde{z}_v^{(l)}||^2 \end{aligned} \quad (10)$$

where $\tilde{z}_v^{(l)}$ is layer-l pre-activation of node v and $\dot{\sigma}(\cdot)$ denotes the derivative of the activation function with respect to $\tilde{z}_v^{(l)}$.

Assumption 3 (Bounded Norm)**.** *The (Frobenius) norm of parameter matrix, propagation vector, and node feature vector are bounded as $||w^{(l)}||_F^2 \leq B_w, \forall l \in [L], ||P_v||^2 \leq B_p, \forall v \in \mathcal{V}, ||x_v||^2 \leq B_x, \forall v \in \mathcal{V}$.*

Lemma 2 (Upper Bound of σ_{bias})**.** *Under Assumptions 2 and 3, $\sigma_{bias} = \max_{t\in[T]} E[||\nabla\mathcal{L}(w_t) - E[\tilde{g}_t]||^2]$ in Theorem 1 is upper bounded by $\frac{1}{K}\sum_{k\in[K]}\mathcal{O}(||\tilde{X}^k - X||_F^2)$.*

All missing proofs can be found in the appendix.

Discussion. Lemma 2 shows that the irreducible bias error in the convergence of update method (4) would diminish to zero if local feature matrix at each worker is complete, *i.e.* $\tilde{X}^k = X$. Further, with the increase of the number of feature dimensions at each worker, the value of this error term decreases. This validates that the error term σ_{bias}, that prevents correct convergence of the GNN model, is attributed to the incomplete feature matrix at each worker.

2) Lower Bound of Gradient Norm: We establish a convergence lower bound of the gradient norm in a strongly convex setting, and demonstrate that while σ_{bias} only appears in the upper bound of the gradient norm in Theorem 1, it indeed prevents the gradient norm from approaching 0 and the model from converging to the optimum of the objective (2). The intrinsic existence of the irreducible error in the convergence of update method (4) is not an artifact of our analysis.

Lemma 3 (Lower Bound of $E||\nabla\mathcal{L}(w_t)||^2$)**.** *Considering a strongly convex optimization objective $\mathcal{L}(w) = \frac{1}{2\mathcal{N}}\sum_{v\in\mathcal{V}}||w - \sum_{u\in\mathcal{N}_1(v)} x_u||^2$, the gradient norm resulting from the update method (4) respects*

$$\lim_{T\to\infty}||\nabla\mathcal{L}(w_T)||^2 = \lim_{T\to\infty}||\frac{1}{K}\sum_{k=1}^{K}\frac{1}{\mathcal{N}}\sum_{v\in\mathcal{V}}\sum_{u\in\mathcal{N}_1(v)}(\tilde{x}_u^k - x_u)||^2$$
$$= ||\nabla\mathcal{L}(w_T) - \tilde{g}_T||^2 = \sigma_{bias} \quad (11)$$

Discussion. In Lemma 3, we derive the exact gradient norm using the update method of (4) without any scaling. The precise gradient norm analysis reveals that the gradient norm is indeed affected by σ_{bias} and fails to approach 0 with arbitrary feature partition method. Even though a neural network is normally non-convex and we can not derive such an exact gradient norm value for it, this result, in the context of a strongly convex objective, provides evidence for the inherent existence of convergence error when each worker uses an incomplete feature vector $\tilde{x}_u^k$.

B. The Case with Cross-worker Feature Fetching

Having obtained an affirmative answer to the first question, we now turn our attention to the second question and explore the frequency of feature fetching during GNN training. Let $\mathcal{T}_{w/o}$ and $\mathcal{T}_w$ denote the set of training iterations without and with cross-worker feature fetching, respectively. Suppose there are a total of Λ training iterations with cross-worker feature fetching, i.e., $|\mathcal{T}_w| = \Lambda$, where parameters of GNN model are updated following (3). Between every two training iterations that involve cross-worker feature fetching, there are several training iterations without cross-worker feature fetching, where the GNN model is updated using (4). Specifically, we use τ_λ to represent the number of training iterations without feature fetching performed between the λ-th and $(\lambda + 1)$-th

[3]Convergence to global optimum with SGD in non-convex optimization cannot be ensured.

model update steps (i.e., training iterations) with cross-worker feature fetching. Then we have the following expressions:

$$\mathcal{T}_{w/o}(\lambda) = \{\lambda + \sum_{j=1}^{\lambda-1} \tau_j + s, s \in [\tau_\lambda]\}, \mathcal{T}_w(\lambda) = \{\lambda + \sum_{j=1}^{\lambda-1} \tau_j\}, \forall \lambda \in [\Lambda]$$

where $\mathcal{T}_{w/o}(\lambda)$ denotes the set of training iterations without cross-worker feature fetching after the λ-th model update with cross-worker feature fetching, and $\mathcal{T}_w(\lambda)$ represents the index of the training iteration corresponding to the λ-th model update step with cross-worker feature fetching. Training iterations with and without cross-worker feature fetching are indexed together as 1, 2, ..., T. The complete set of training iterations $\mathcal{T} = \mathcal{T}_w \cup \mathcal{T}_{w/o}$, with $\mathcal{T}_w = \mathcal{T}_w(1) \cup ... \cup \mathcal{T}_w(\Lambda)$ and $\mathcal{T}_{w/o} = \mathcal{T}_{w/o}(1) \cup ... \cup \mathcal{T}_{w/o}(\Lambda)$. In training iteration t, model parameters are updated using (3) if $t \in \mathcal{T}_w$, or using (4) if $t \in \mathcal{T}_{w/o}$. We refer to this as distributed GNN training with *periodic feature fetching*. Solely based on assumption 1, we derive the following convergence result for it.

Theorem 4 (Convergence of Periodic Feature Fetching)**.** *Feature-partition based distributed GNN training with periodic feature fetching converges as long as:*

$$\tau_\lambda \leq \frac{(1-\eta L_f)E[||\nabla\mathcal{L}(w_{t\in\mathcal{T}_w(\lambda)})||^2]}{\sigma_{bias}}, \forall \lambda \in [\Lambda] \quad (12)$$

Proof. We give the proof sketch here and leave the full proof to appendix. The main idea is to analyze the convergence behavior of the GNN model during the two sets of training iterations, $\mathcal{T}_{w/o}$ and $\mathcal{T}_w$, so that the convergence of GNN model for each training iteration $t \in \mathcal{T} = \mathcal{T}_w \cup \mathcal{T}_{w/o}$ can be derived as follows:

$$\sum_{t\in\mathcal{T}} E[||\nabla\mathcal{L}(w_t)||^2] \leq \frac{2}{\eta}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \sum_{t\in\mathcal{T}_{w/o}} \sigma_{bias} + \sum_{t\in\mathcal{T}_{w/o}} \eta L_f \sigma_{var} + \sum_{t\in\mathcal{T}_w} \eta L_f \hat{\sigma}_{var} + \sum_{t\in\mathcal{T}_w} (\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2] \quad (13)$$

where $\hat{\sigma}_{var}$ is the variance upper bound of $E[||\nabla\mathcal{L}(w_t) - g_t||^2]$ with $g_t = \frac{1}{K}\sum_{k\in[K]} \frac{1}{|B_t^k|}\sum_{v\in B_t^k} \nabla\mathcal{L}(h_v^{(L)}, w_t)$, which is an unbiased estimator of $\nabla\mathcal{L}(w_t)$, and $\hat{\sigma}_{var} = 0$ if $B_t^k = \mathcal{V}$. The last term $\sum_{t\in\mathcal{T}_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2]$ in the above result is less than 0 due to the requirement of $\eta \leq \frac{1}{L_f}$ in the convergence result of the GNN model in $t \in \mathcal{T}_{w/o}$ (Theorem 1), then we could *remove the non-vanishing bias error and ensure model convergence by adjusting the value of τ_λ to make $\sum_{t\in\mathcal{T}_{w/o}} \sigma_{bias}$ and $\sum_{t\in\mathcal{T}_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2]$ cancel out each other, i.e.,* $\sum_{t\in\mathcal{T}_{w/o}} \sigma_{bias} + \sum_{t\in\mathcal{T}_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2] \leq 0$, which gives us the solution to τ_λ in (12). Then the subsequent convergence result follows:

$$\min_{t\in T} E[||\nabla\mathcal{L}(w_t)||^2] \leq \frac{1}{T}\sum_{t\in\mathcal{T}} E[||\nabla\mathcal{L}(w_t)||^2]$$
$$\overset{(a)}{\leq} \frac{2}{\eta T}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \frac{\eta L_f}{T}(\sum_{t\in\mathcal{T}_{w/o}} \sigma_{var} + \sum_{t\in\mathcal{T}_w} \hat{\sigma}_{var}) \quad (14)$$
$$\overset{(b)}{\leq} \frac{2}{\sqrt{T}}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \frac{L_f}{\sqrt{T}}\sigma \overset{T\to\infty}{=} 0 \qquad \square$$

where (a) holds because $\sum_{t\in\mathcal{T}_{w/o}} \sigma_{bias} + \sum_{t\in\mathcal{T}_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2] \leq 0$ and (b) is valid when $\sigma = \max(\sigma_{var}, \hat{\sigma}_{var})$ and $\eta = \frac{1}{\sqrt{T}}$.

Remark 1 Although cross-worker feature fetching is unavoidable, we offer *a positive response* to the second question. Feature fetching in every training iteration is superfluous; as long as the number of training iterations without feature fetching between two consecutive model update steps with cross-worker feature fetching is no larger than $(1 - \eta L_f)E[||\nabla\mathcal{L}(w_{t\in\mathcal{T}_w(\lambda)})||^2]/\sigma_{bias}$, the GNN model is ensured to converge to the stationary point of the objective (2).

Remark 2 The derived solution to τ_λ in (12) indicates that to ensure model convergence, a larger bias σ_{bias} due to incomplete local features allows less model update steps without cross-worker feature fetching. It also implies that during distributed GNN training, the frequency of cross-worker feature fetching (indicated by τ_λ, the number of model updates without feature fetching) should increase (τ_λ should decrease) over time, as the gradient norm in the upper bound of τ_λ typically follows a non-increasing trend. This aligns well with intuition: as training gradually approaches convergence/stationary point, even small biases can have a significant impact on the optimization dynamics, necessitating more frequent cross-worker feature fetching to maintain accurate and reliable updates.

IV. Feature Communication Plan

Built upon the insights obtained in Sec. III, we devise an optimal feature fetching communication plan. Intuitively, frequent feature fetching can reduce the bias and aid model convergence, while increasing the communication time. We identify an optimal feature fetching frequency to strike the right balance and minimize model training convergence time.

We establish a relationship between the model convergence error and training runtime under our periodic feature fetching, using a similar approach as in [25]. We consider every training iteration with cross-worker feature fetching is followed by τ subsequent training iterations without cross-worker feature fetching. Following the convergence result in (13), the model convergence under this feature fetching frequency setting is:

$$\frac{1}{T}\sum_{t\in\mathcal{T}} E[||\nabla\mathcal{L}(w_t)||^2] \leq \frac{1}{T}\{\frac{2}{\eta}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \Lambda\tau\sigma_{bias} + \Lambda\tau\eta L_f\sigma_{var} + \Lambda\eta L_f\hat{\sigma}_{var} + \sum_{t\in\mathcal{T}_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2]\} \quad (15)$$

The convergence error on the right hand side (RHS) of (15) is related to the total number of training iterations. We further convert the iteration number T into the total training runtime. We use $\bar{C}$ and $\mathcal{T}_{commu} = \frac{\bar{B}d}{Kb}$ to denote the average per-mini-batch model computation time and the average communication time for each worker's feature fetching in one training iteration, respectively, where $\bar{B}$ is the average number of nodes involved in the subgraphs of a mini-batch and b represents the inter-worker bandwidth. We use the dimension number of feature vectors in communication time computation, assuming each dimension using a constant number of bytes. We can establish the relation between the number of training iterations T and the training runtime C_T as

$$C_T = T(\bar{C} + \frac{\mathcal{T}_{commu}}{\tau + 1}) \approx T(\bar{C} + \frac{\mathcal{T}_{commu}}{\tau}) = T(\bar{C} + \frac{\bar{B}d}{\tau K b}) \quad (16)$$

Here we omit the communication time for gradient sychronization in each training iterations, as the GNN parameter size is typically much smaller than feature size.

By substituting T in the RHS of (15) by $C_T/(\bar{C} + \frac{\bar{B}d}{\tau Kb})$ according to (16), we derive the GNN convergence error with respect to training runtime C_T:

$$\frac{1}{T}\sum_{t\in\mathcal{T}} E[||\nabla\mathcal{L}(w_t)||^2] \leq C + \frac{\bar{C}\boldsymbol{\tau}}{C_T}\{\Lambda\sigma_{bias} + \Lambda\eta L_f\sigma_{var}\} + \frac{\bar{B}d}{\boldsymbol{\tau}KbC_T}$$
$$\{\frac{2}{\eta}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \Lambda\eta L_f\hat{\sigma}_{var} + \sum_{t\in T_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2]\} \tag{17}$$

where $C = \frac{\bar{C}}{C_T}(\frac{2}{\eta}(\mathcal{L}(w_1) - \mathcal{L}(w^*)) + \Lambda\eta L_f\hat{\sigma}_{var} + \sum_{t\in T_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2]) + \frac{\bar{B}d}{KbC_T}(\Lambda\sigma_{bias} + \Lambda\eta L_f\sigma_{var})$, a constant term independent of τ. The convergence error in (17) aligns with our intuitions, as frequent cross-worker feature fetching (smaller τ) introduces less bias (smaller coefficient before σ_{bias}), but incurs longer communication time (larger $\frac{\bar{B}d}{\tau Kb}$). **Thus striking the optimal trade-off between convergence error and feature communication time is to find the optimal τ^* minimizing both terms at the same time, which is equivalent to minimizing the RHS of (17).** The convexity of (17) on the feature fetching frequency τ (bold in (17)) makes it possible to find an optimal solution of τ, that achieves the best trade-off between convergence error and feature communication time. By setting the derivative of (17) with respect to τ to 0, we derive the optimal feature fetching frequency as follows:

$$\tau^* = \sqrt{\frac{\bar{B}d(\frac{2}{\eta}\mathcal{L}(w_1) + \Lambda\eta L_f\hat{\sigma}_{var} + \sum_{t\in T_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(w_t)||^2])}{\bar{C}Kb(\Lambda\sigma_{bias} + \Lambda\eta L_f\sigma_{var})}} \tag{18}$$

Here we assume $\mathcal{L}(w^*) = 0$, which is reasonable since the loss value gets close to 0 when the model converges in most training scenarios (see training curves in Fig. 4 and Fig. 5).

The optimal τ^* involves $\sum_{t\in T_w}(\eta L_f - 1)E[||\nabla\mathcal{L}(W_t)||^2]$, the gradient norm of all training iterations with cross-worker feature fetching, which cannot be known without completing the entire training process. To circumvent the need of computing gradient norm of all training iterations, we treat the training, starting from the λ-th ($\forall\lambda \in [\Lambda]$) model update step with cross-worker feature fetching, as a training re-start from model parameter w_{t_λ}. The re-started training between the λ-th and $\lambda + 1$-th model update step with cross-worker feature fetching follows the same convergence property as (15) during the training interval with $\Lambda = 1$ (there is only one training iteration with cross-worker feature fetching during the re-started training, which begins from the λ-th model update step with cross-worker feature fetching and continues with τ_λ model update steps without cross-worker feature fetching). Thus the optimal τ_λ^* follows the same form as (18) except $\Lambda = 1$ and $|\mathcal{T}_w| = 1$. We derive the following optimal feature fetching frequency to use starting from the λ-th model update step, by replacing the sum of gradient norm of all training iterations with cross-worker feature fetching in (18) by $E[||\nabla\mathcal{L}(W_{t_\lambda})||^2]$, since t_λ is the only model update step with cross-worker feature fetching during this re-started training:

$$\tau_\lambda^* = \sqrt{\frac{\bar{B}d(\frac{2}{\eta}\mathcal{L}(w_{t_\lambda}) + \eta L_f\hat{\sigma}_{var} + (\eta L_f - 1)E[||\nabla\mathcal{L}(w_{t_\lambda})||^2])}{\bar{C}Kb(\sigma_{bias} + \eta L_f\sigma_{var})}} \tag{19}$$

This optimal feature fetching frequency τ_λ^* achieves the best trade-off between model error and feature communication time for the training between λ-th and $\lambda + 1$-th model update steps with cross-worker feature fetching.

While the above τ_λ^* ensures the right trade-off in each such training interval, it does not necessarily guarantee the minimum gradient norm will converge to 0 over time. Thus we combine our analysis result of $\tau_\lambda \leq \frac{(1-\eta L_f)E[||\nabla\mathcal{L}(w_{t_\lambda})||^2]}{\sigma_{bias}}$ in Sec. III-B, which ensures a minimum gradient norm value of 0 for GNN models, and obtain the optimal feature fetching frequency to use starting from the λ-th model update step with cross-worker feature fetching by ensuring that the optimal solution to τ_λ^* at least guarantees the model convergence.

$$\tau_\lambda^* = \min\left\{\begin{array}{c}\frac{(1-\eta L_f)E[||\nabla\mathcal{L}(w_{t_\lambda})||^2]}{\sigma_{bias}}, \\ \sqrt{\frac{\bar{B}d(\frac{2}{\eta}\mathcal{L}(w_{t_\lambda})+\eta L_f\hat{\sigma}_{var}+(\eta L_f-1)E[||\nabla\mathcal{L}(w_{t_\lambda})||^2])}{\bar{C}Kb(\sigma_{bias}+\eta L_f\sigma_{var})}}\end{array}\right\} \tag{20}$$

The complete training process is in Alg. 1.

Algorithm 1 Distributed GNN Training with Optimal Feature Fetching Plan (perspective of each worker $k \in [K]$)

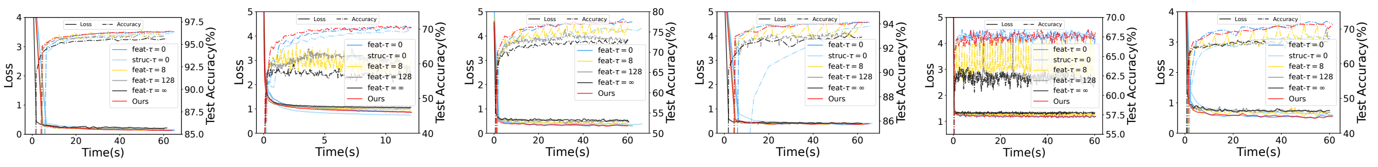
INPUT: Adjacency matrix $A \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$, feature matrix $X^k \in \mathbb{R}^{\mathcal{N}\times\frac{d}{K}}$, initial model parameters w_1, number of training iterations T
OUTPUT: Model parameter w_T

```
1:  Set number of model updates with feature fetching to $\lambda = 1$, training iteration number to $t = 1$;
2:  while $t \leq T$ do
3:      Calculate $\tau_\lambda^*$ based on (20) and $w_t$;
4:      Sample mini-batch $B_t^k$ and associated subgraphs $\mathcal{G}_t^k$;
5:      Fetch features for nodes $u \in \mathcal{G}_t^k$ and recover $x_u \in \mathbb{R}^d$;
6:      Compute $h_v^{(L)}, \forall v \in B_t^k$ with (1) and $x_u, \forall u \in \mathcal{G}_t^k$;
7:      Update GNN model with (3) to obtain $w_{t+1}$;
8:      $t = t + 1$;
9:      for $\zeta \in [\tau_\lambda^*]$ do
10:         Sample mini-batch $B_t^k$ and associated subgraphs $\mathcal{G}_t^k$;
11:         Compute $\tilde{h}_v^{(L)}(k), \forall v \in B_t^k$ with (1) and $\tilde{x}_u^k, \forall u \in \mathcal{G}_t^k$
12:         Update GNN model with (4) to obtain $w_{t+1}$;
13:         $t = t + 1$;
14:     end for
15:     $\lambda = \lambda + 1$;
16: end while
```

Remark 1 The value of τ_λ^* can be efficiently calculated: (1) L_f can be estimated according to the property of Lipschitz smoothness [26]. (2) $\bar{B}, \bar{C}, \sigma_{var}, \sigma_{bias}$, and $\hat{\sigma}_{var}$ can be inferred based on their definitions, by calculating the respective terms from several trials of training iterations before actual training starts. (3) All other terms are either constant, such as d, η, or can be obtained during the training process, including $\mathcal{L}(w_{t_\lambda})$ and $\nabla\mathcal{L}(w_{t_\lambda})$.

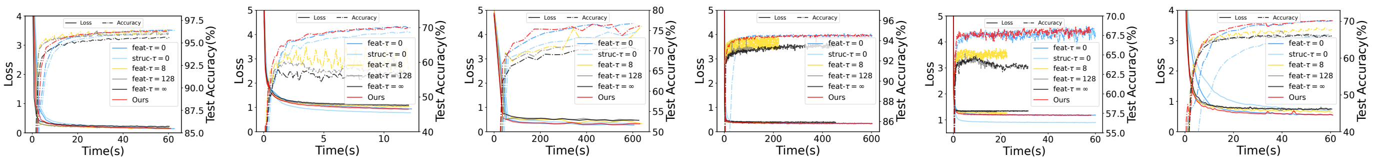
TABLE II: Training Time to Reach a Target Accuracy (seconds). × indicates the speedup obtained by dividing the convergence time of each baseline by the convergence time of our method. / means the respective baseline converges to an accuracy lower than the target accuracy.

Dataset		Reddit		ogbn-arxiv		ogbn-products	
Model (Target Test Accuracy)		GraphSage(96%)	GAT(94%)	GraphSage(70%)	GAT(68%)	GraphSage(75%)	GAT(70%)
With neighbor Sampling	feat-(τ=0)	25.95 (1.07×)	32.70 (0.94×)	4.54 (1.60×)	3.72 (0.51×)	14.99 (1.34×)	23.46 (0.67×)
	struc-(τ=0)	38.92 (1.60×)	/	9.14 (3.21×)	10.19 (1.41×)	47.09 (4.20×)	/
	feat-(τ=∞)	/	/	/	/	/	/
	feat-(τ=8)	12.46 (0.51×)	45.91 (1.15×)	/	/	12.05 (1.07×)	39.04 (1.11×)
	feat-(τ=128)	50.98 (2.10×)	60.57 (1.52×)	/	/	32.10 (2.86×)	58.80 (1.67×)
	Ours	24.27	39.78	2.85	7.23	11.2	35.17
Without neighbor Sampling	feat-(τ=0)	29.86 (1.12×)	38.84 (0.97×)	7.22 (1.18×)	13.79 (1.57×)	330.00 (1.90×)	53.21 (1.01×)
	struc-(τ=0)	45.05 (1.68×)	/	/	13.53 (1.54×)	/	/
	feat-(τ=∞)	/	/	/	/	/	/
	feat-(τ=8)	13.63 (0.51×)	22.19 (0.55×)	/	/	343.87 (1.98×)	/
	feat-(τ=128)	28.52 (1.07×)	60.47 (1.51×)	/	/	/	/
	Ours	26.76	40.07	6.10	8.76	173.91	52.88



(a) Reddit(GraphSage) (b) Arxiv(GraphSage) (c) Products(GraphSage) (d) Reddit(GAT) (e) Arxiv(GAT) (f) Products(GAT)

Fig. 4: GNN Training Convergence with Neighbor Sampling



(a) Reddit(GraphSage) (b) Arxiv(GraphSage) (c) Products(GraphSage) (d) Reddit(GAT) (e) Arxiv(GAT) (f) Products(GAT)

Fig. 5: GNN Training Convergence without Neighbor Sampling

V. Performance Evaluation

A. Methodology

Settings. We implement distributed GNN training over four fully-connected machines, using Pytorch 1.13.1 [27] on DGL 1.1.1 [17]. Each machine is equipped with one NVIDIA RTX 3090 Ti GPU, one Intel i9-12900KS CPU and 126 GB host memory. The default bandwidth between each pair of workers is 10Gbps and the mini-batch size at each worker is 1024.

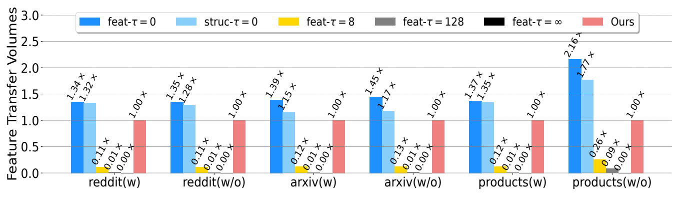
GNN Models and Datasets. We train two representative GNN models, GraphSage [2] (two layers with hidden size 256 and the mean aggregator) and GAT [28] (two layers with hidden size 256 and 4 heads of dropout 0.5), on three large graph datasets: the reddit dataset (0.23 million nodes, 114.61 million edges, and 602-dimensional node features), the ogbn-arxiv dataset (0.16 million nodes, 1.16 million edges, and 128-dimensional node features), and the ogbn-products dataset (2.44 million nodes, 61.85 million edges, and 100-dimensional node features) [12]. In experiments with neighborhood sampling, we adopt uniform neighbor sampling [2] with a fanout (number of neighbor nodes to be sampled at each layer) of 10-15. The learning rate is set to 0.005 in all experiments.

Baselines. We compare our design with four baselines. (1) **feat-(τ=0)**: feature-partition based distributed GNN training with cross-worker feature fetching in every training iteration. (2) **struc-(τ=0)**: structure-partition based distributed GNN training with cross-worker subgraph sampling and feature fetching in every iteration; nodes in a graph dataset are evenly distributed among all workers. (3) **feat-(τ=∞)**: feature-partition based distributed training without cross-worker feature fetching during the whole training process; the only information exchanged among workers are gradients calculated based on local incomplete features. (4) **feat-(τ=#)**: feature-partition based distributed training with fixed feature fetching frequency, where one model update step with cross-worker feature fetching is performed after every fixed number (specified by #) of training iterations without feature fetching.

B. GNN Training Convergence

We first evaluate the convergence behavior of GNN training with neighborhood sampling and without (complete L-hop neighbor nodes are used). Table II records the model convergence time required to reach a target test accuracy. Fig. 4 and Fig. 5 plot the training convergence curves.

The following observations are made: (1) GNN training with only gradients exchanged among workers (**feat-(τ=∞)**) leads to the largest training loss and the lowest test accuracy, consistent with our theoretical results on the non-vanishing bias error in this case in Sec. III-A. (2) In feature-partition based distributed training, as the feature fetching frequency decreases (larger τ), test accuracy of GNN models decreases, verifying our theoretical result in Sec. III-B (the smaller τ is, the more feasible for it is to cancel out the irreducible error). (3) Our optimal feature fetching frequencies leads to an average 1.5× training speed-up as compared to the baselines, reducing about 30% convergence time to the target accuracy. The speedup can be further improved when overlapping model computation and feature fetching, as feature fetching is only


(a) GraphSage Model

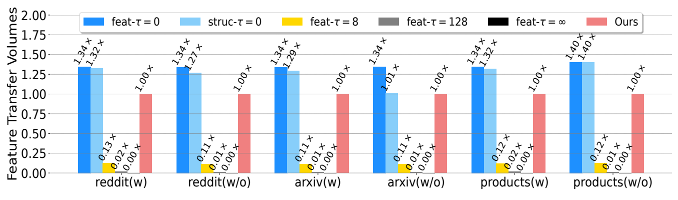

(b) GAT Model

Fig. 6: Feature Data Transfer Size Comparison. (w) and (w/o) denote training with and without neighbor sampling, respectively.

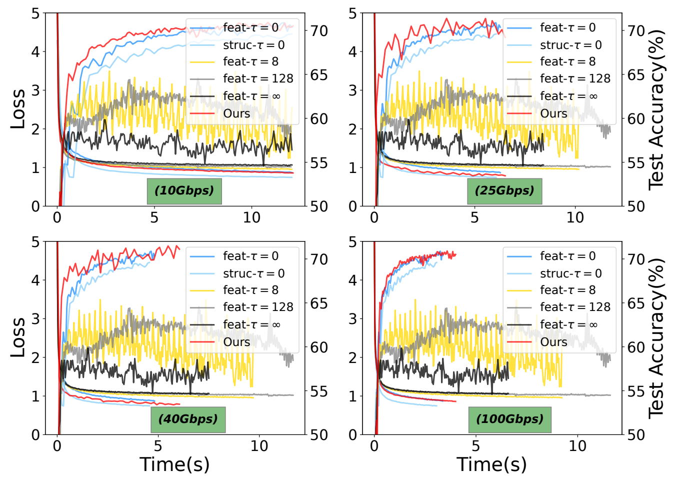


Fig. 7: GNN Training Convergence: diff. bandwidth among workers

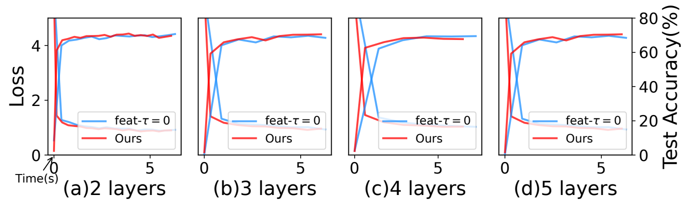


Fig. 8: GNN Training Convergence: diff. numbers of model layers

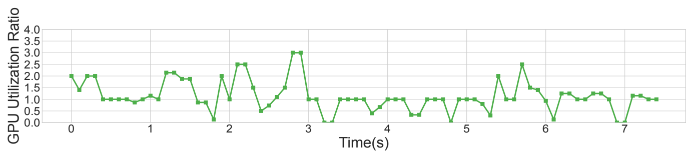


Fig. 9: GPU Utilization: ratio between our method and **feat-τ=0**

periodically performed in our method. *Although our frequencies are theoretically optimal, the presence of estimation errors in computing the optimal solution τ^*_λ may affect its performance*: our method outperforms more than 85% of the different cases that we evaluated, and achieves a comparable test accuracy as feature fetching in every iteration (**feat-(**τ=0**)**). (4) As compared to structure-partition based training, feature-partition based training achieves faster convergence due to eliminating cross-worker subgraph sampling; the advantages are more evident when considering the additional graph partitioning time required by structure partitioning.

C. Communication Overhead

We next compare the average feature data transfer volumes among all workers per training iteration. The data transfer size is normalized against the feature data transfer volume of our method. Fig. 6a and Fig. 6b show that the communication overhead for cross-worker feature fetching using our method is consistently lower than that of feature fetching in every iteration (**feat-(**τ=0) and **struc-(**τ=0)), with an average reduction of more than 25%. The value of τ^*_λ computed by our method has initial value of several dozens, and gradually decreases over time during the training process. For example, when training GraphSage on obgn-products, τ^*_λ starts at 18, and gradually decreases and becomes 0 eventually.

D. Impact of Bandwidth between Workers

We train the GraphSage model on ogbn-arxiv dataset with neighbor sampling, using different bandwidth levels between workers. With a higher bandwidth level, we expect the optimal τ^*_λ to decrease (cross-worker feature fetching frequency to increase), as the communication overhead of cross-worker feature fetching is less significant with higher bandwidth. This trend can be observed from the training convergence curves in Fig. 7: when the bandwidth between workers is larger, our method derives smaller τ^*_λ, approaching the performance of **feat-(**τ=0) (feature fetching in every iteration).

E. Impact of the Number of GNN Layers

We train the GraphSage model on ogbn-arxiv dataset with 10Gbps inter-worker bandwidth with neighbor sampling, and vary the number of layers in the GNN model. Fig. 8 plots the convergence curves of distributed GNN training using our method and **feat-(**τ=0**)**. Our method exhibits consistent improvement of convergence speed with more GNN layers, different from neighbor sampling and caching, whose benefits become marginal as the number of GNN layers increases [29] [30]. This is because feature transfer volumes per iteration increase exponentially with the number of GNN layers; omitting some feature fetching steps brings more benefits.

F. GPU Utilization

We train the GraphSage model on the ogbn-products dataset and record the ratio of GPU utilization between our method and (**feat-τ=0**). The results in Fig. 9 show that our method enhances GPU utilization, starting with a 2-3$\times$ improvement, which gradually approaches 1 as τ^*_λ approaches 0 over time.

VI. Conclusion

This paper focuses on feature fetching communication reduction in feature-partition based distributed GNN training. We thoroughly study the convergence behaviour of GNN models and design an optimal feature fetching communication plan. We evaluate our design extensively and demonstrate its effectiveness.

References

[1] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proceedings of the International Conference on Learning Representations*, 2016.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[3] X. Li, Y. Shang, Y. Cao, Y. Li, J. Tan, and Y. Liu, "Type-aware Anchor Link Prediction across Heterogeneous Networks based on Graph Attention Network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 01, 2020, pp. 147–155.

[4] S. Yun, S. Kim, J. Lee, J. Kang, and H. J. Kim, "Neo-gnns: Neighborhood Overlap-aware Graph Neural Networks for Link Prediction," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13 683–13 694, 2021.

[5] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A Fair Comparison of Graph Neural Networks for Graph Classification," in *Proceedings of the International Conference on Learning Representations*, 2019.

[6] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning Discrete Structures for Graph Neural Networks," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2019, pp. 1972–1982.

[7] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph Convolutional Neural Networks for Web-scale Recommender Systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 974–983.

[8] Z. Cai, X. Yan, Y. Wu, K. Ma, J. Cheng, and F. Yu, "DGCL: an Efficient Communication Library for Distributed GNN Training," in *Proceedings of the Sixteenth European Conference on Computer Systems*, 2021, pp. 130–144.

[9] 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," in *Proceedings of the 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 2020, pp. 36–44.

[10] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with roc," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 187–198, 2020.

[11] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[12] 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," *Advances in Neural Information Processing Systems*, vol. 33, pp. 22 118–22 133, 2020.

[13] T. Liu, Y. Chen, D. Li, C. Wu, Y. Zhu, J. He, Y. Peng, H. Chen, H. Chen, and C. Guo, "BGL: GPU-Efficient GNN Training by Optimizing Graph Data I/O and Preprocessing," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI'23)*, 2023, pp. 103–118.

[14] S. Gandhi and A. P. Iyer, "P3: Distributed Deep Graph Learning at Scale," in *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*, 2021, pp. 551–568.

[15] T. Kaler, A. Iliopoulos, P. Murzynowski, T. Schardl, C. E. Leiserson, and J. Chen, "Communication-Efficient Graph Neural Networks with Probabilistic Neighborhood Expansion Analysis and Caching," *Proceedings of Machine Learning and Systems*, vol. 5, 2023.

[16] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, "Graph Neural Networks in Recommender Systems: A Survey," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–37, 2022.

[17] 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," *arXiv preprint arXiv:1909.01315*, 2019.

[18] Euler Graph Library. [Online]. Available: https://github.com/alibaba/euler

[19] H. Chen, M. Liu, Y. Zhao, X. Yan, D. Yan, and J. Cheng, "G-miner: an Efficient Task-oriented Graph Mining System," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–12.

[20] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," in *Proceedings of the 10th USENIX symposium on Operating Systems Design and Implementation (OSDI'12)*, 2012, pp. 17–30.

[21] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "GraphX: Graph Processing in a Distributed Dataflow Framework," in *Proceedings of the 11th USENIX symposium on operating systems design and implementation (OSDI'14)*, 2014, pp. 599–613.

[22] J. Perazzone, S. Wang, M. Ji, and K. S. Chan, "Communication-efficient Device Scheduling for Federated Learning using Stochastic Optimization," in *Proceedings of the IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1449–1458.

[23] H. Yu, S. Yang, and S. Zhu, "Parallel Restarted SGD with Faster Convergence and Less Communication: Demystifying Why Model Averaging Works for Deep Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 5693–5700.

[24] J. Chen, J. Zhu, and L. Song, "Stochastic Training of Graph Convolutional Networks with Variance Reduction," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2018, pp. 942–950.

[25] B. Du and C. Wu, "Federated Graph Learning with Periodic Neighbour Sampling," in *Proceedings of the 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. IEEE, 2022, pp. 1–10.

[26] M. Fazlyab, A. Robey, H. Hassani, M. Morari, and G. Pappas, "Efficient and Accurate Estimation of Lipschitz Constants for Deep Neural Networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[27] 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," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[28] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," in *Proceedings of the International Conference on Learning Representations*, 2018.

[29] Y. Bai, C. Li, Z. Lin, Y. Wu, Y. Miao, Y. Liu, and Y. Xu, "Efficient Data Loader for Fast Sampling-based GNN Training on Large Graphs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 10, pp. 2541–2556, 2021.

[30] 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*, 2020, pp. 401–415.