PipeGCN: Efficient Full-Graph Training of Graph Convolutional Networks with Pipelined Feature Communication

Anonymous authors
Paper under double-blind review

Abstract

Graph Convolutional Networks (GCNs) is the state-of-the-art method for learning graph-structured data, and training large-scale GCNs requires distributed training across multiple accelerators such that each accelerator is able to hold a partitioned subgraph. However, distributed GCN training incurs prohibitive overhead of communicating node features and feature gradients among partitions for every GCN layer in each training iteration, limiting the achievable training efficiency and model scalability. To this end, we propose PipeGCN, a simple-yet-effective scheme that hides the communication overhead by pipelining inter-partition communication with intra-partition computation. It is non-trivial to pipeline for efficient GCN training, as communicated node features/gradients will become stale and thus can harm the convergence, negating the pipeline benefit. Notably, little is known regarding the convergence rate of GCN training with both stale features and stale feature gradients. This work not only provides a theoretical convergence guarantee but also finds the convergence rate of PipeGCN to be close to that of the vanilla distributed GCN training without staleness. Furthermore, we develop a smoothing method to further improve PipeGCN's convergence. Extensive experiments show that PipeGCN can largely boost training throughput (up to 2.2$\times$) while achieving the same accuracy as its vanilla counterpart and outperforming existing full-graph training methods. All code will be released publicly upon acceptance.

1 Introduction

Graph convolutional networks (GCNs) (Kipf & Welling, 2016) have gained great popularity recently as they demonstrated the state-of-the-art (SOTA) performance in various graph-based learning tasks, including node classification (Kipf & Welling, 2016), link prediction (Zhang & Chen, 2018), graph classification (Xu et al., 2018), and recommendation systems (Ying et al., 2018). The promising performance of GCNs is due to their diverse neighborhood connectivity, which provides a greater applicability to graph-based data than convolutional neural networks (CNNs) that adopt a fixed regular neighborhood structure. In particular, a GCN aggregates all features from the neighbor node set for a given node, the feature of which is then updated via a multi-layer perceptron. Such a two-step process (i.e., *aggregate* and *update*) empowers GCNs to better learn graph structures.

However, training GCNs at scale has been a challenging problem, as a prohibitive amount of compute and memory resources are required to train a real-world large-scale graph, let alone exploring deeper and more advanced models. To tackle this challenge, various sampling-based methods have been proposed to reduce the resource requirement at a cost of incurring feature approximation errors. A straightforward instance is to create mini-batches by sampling neighbors (e.g., GraphSAGE (Hamilton et al., 2017) and VR-GCN (Chen et al., 2018)) or to extract subgraphs as training samples (e.g., Cluster-GCN (Chiang et al., 2019) and GraphSAINT (Zeng et al., 2020)).

In addition to sampling-based methods, distributed GCN training has emerged as a promising alternative, as it enables large *full graph* training of GCNs across multiple accelerators such as GPUs. The essence of this approach is to separate a giant graph into several small partitions, each of which is able to fit into a single GPU, and then train these subgraphs locally on GPUs but with indispensable communication. Following this direction, several recent works (Ma et al., 2019; Jia et al., 2020; Tripathy et al., 2020; Thorpe et al., 2021) have been proposed and verified the great potential of distributed GCN training. P^3 (Gandhi & Iyer, 2021) follows another direction that splits the data along the feature dimension and leverages intra-layer model parallelism for training, which shows superior performance on small models.

In this work, we propose a new method of distributed GCN training, PipeGCN, which targets achieving a full-graph accuracy with boosted training efficiency. Our main contributions are:

- We first analyze two efficiency bottlenecks in distributed GCN training: *significant communication overhead* and *frequent synchronization*, and then propose a simple-yet-effective technique called PipeGCN to address the above two bottlenecks by pipelining inter-partition communication with intra-partition computation to hide the communication overhead.
- We address the challenge raised by PipeGCN, i.e., staleness in communicated features and feature gradients, by providing a theoretical convergence guarantee of PipeGCN which finds the convergence rate to be $\mathcal{O}(T^{-\frac{2}{3}})$, i.e., close to vanilla distributed GCN training without staleness. *To the best of our knowledge, this is the first work providing theoretical proof for the convergence of GCN training with* ***both*** *stale feature and stale feature gradients.*
- We further propose a low-overhead smoothing method to further improve PipeGCN's convergence by reducing the error incurred by the staleness.
- Extensive empirical and ablation studies consistently validate the advantages of PipeGCN over both vanilla distributed GCN training (boosting training throughput by up to 2.2× while achieving the same or a better accuracy) as well as SOTA full-graph training methods.

2 Background and Related Works

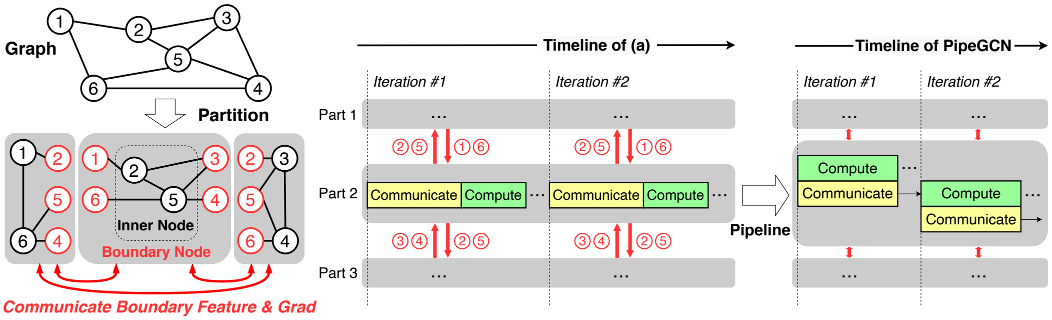
Graph Convolutional Networks. GCNs (Kipf & Welling, 2016) exhibit a powerful learning ability for graph-structured data. They represent each node in a graph as a feature (embedding) vector and learn the feature vector via a two-step process (*neighbor aggregate* and then *update*) for each layer, which can be mathematically described as:

$$z_v^{(\ell)} = \zeta^{(\ell)}\left(h_u^{(\ell-1)} \mid u \in \mathcal{N}(v)\right) \tag{1}$$

$$h_v^{(\ell)} = \phi^{(\ell)}\left(z_v^{(\ell)}, h_v^{(\ell-1)}\right) \tag{2}$$

where $\mathcal{N}(v)$ is the neighbor set of node v in the graph, $h_u^{(\ell)}$ represents the learned embedding vector of node u at the ℓ-th layer, $z_v^{(\ell)}$ is an intermediate aggregated feature calculated by an aggregation function $\zeta^{(\ell)}$, and finally $\phi^{(\ell)}$ is the function for updating the feature of node v. The original GCN (Kipf & Welling, 2016) uses a mean aggregator for $\zeta^{(\ell)}$ and the update function $\phi^{(\ell)}$ is a single-layer perceptron $\sigma(W^{(\ell)}z_v^{(\ell)})$ where $\sigma(\cdot)$ is a non-linear activation function and $W^{(\ell)}$ is the weight matrix. Another famous GCN instance is GraphSAGE (Hamilton et al., 2017) in which $\phi^{(\ell)}$ is $\sigma\left(W^{(\ell)} \cdot \text{CONCAT}\left(z_v^{(\ell)}, h_v^{(\ell-1)}\right)\right)$.

Distributed Training for GCNs. Real-world graphs may contain hundreds of millions of nodes and billions of edges (Hu et al., 2020), for which a training feasible approach is to partition it into small subgraphs (to fit each GPU's resource), and train them in parallel, during which necessary communication is performed to exchange boundary node features and gradients to satisfy GCNs'*neighbor aggregation* (Equ. 1). Such approach is called *vanilla partition-parallel training* and illustrated in Fig. 1 (a). Following this approach, pioneer works have been proposed recently. ROC (Jia et al., 2020), NeuGraph (Ma et al., 2019) and AliGraph (Zhu et al., 2019) perform the partition-parallel training but rely on CPU storage for all partitions and repeated swapping of a partial partition to GPUs. Inevitably, prohibitive CPU-GPU swaps are incurred, plaguing the achievable training efficiency. CAGNET (Tripathy et al., 2020) is different in that it further splits node feature vectors into tiny sub-vectors, which however demands broadcast of those sub-vectors per node and computing them sequentially, thus requires redundant communication and frequent synchronization. More recently, P^3 (Gandhi & Iyer, 2021) proposes to split feature dimension and partition the first layer for parallel training with mitigated communication overhead, but it is based on a strong assumption that the hidden dimensions of a GCN should be considerably smaller than that of input features. A concurrent work Dorylus (Thorpe et al., 2021) proposes to build a fine-grain pipeline along each compute operation in GCN training and supports asynchronous usage of stale features. However, staleness of



(a) Vanilla partition-parallel training (b) Timeline of vanilla partition-parallel training (c) **PipeGCN**

Figure 1: An illustration comparison between vanilla partition-parallel training and PipeGCN.

feature gradients is neither analyzed nor considered for convergence proof, let alone error reduction methods for the incurred staleness.

Table 1: Comparison with Pipe-SGD.

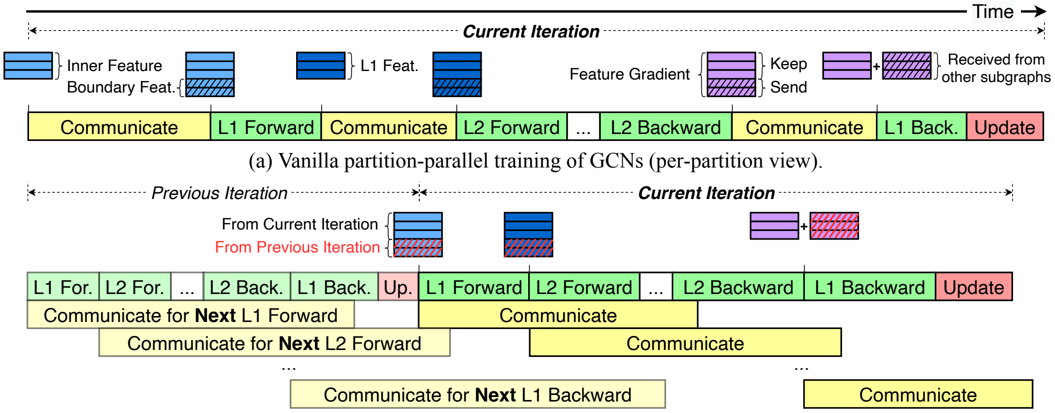
Method	Pipe-SGD	**PipeGCN**
Target	Large Model, Small Feature	Large Feature
Staleness	Weight Gradients	Features and Feature Gradients
Reduce Overhead	AllReduce of Weight Gradient	Aggregation of Feature/Feature Grad.
Convergence Rate	$\mathcal{O}(T^{-\frac{1}{2}})$	$\mathcal{O}(T^{-\frac{2}{3}})$

Asynchronous Distributed Training. Many prior works have been proposed for asynchronous distributed training of DNNs, such as Hogwild! (Niu et al., 2011), SSP (Ho et al., 2013), and MXNet (Li et al., 2014). Most are based on the parameter server architecture with multiple workers running asynchronously to hide communication overhead of each other, at a cost of using stale *weight gradients* from previous iterations. Similarly, other works like Pipe-SGD (Li et al., 2018) pipeline communication with local computation of each worker, trading staleness of *weight gradients* for a better training efficiency (see details in Tab. 1). Nonetheless, these works are for large models with small data, where communication overhead of model weights/gradients are substantial but data feature communications are marginal, if not none. Besides, most asynchronous DNN training focus on convergence with stale *weight gradients* of models, rather than stale *features/feature gradients*. Another direction is to partition a large model along its layers across multiple GPUs and then stream in the data batch through the layer pipeline, e.g., PipeDream (Harlap et al., 2018) and PipeMare (Yang et al., 2021). However, it is also designed for large models with small data and thus not well suited for GCNs. *In a nutshell, little effort has been made to study pipelined training or asynchronous distributed training of GCNs, where feature communication is the major overhead , let alone corresponding theoretical convergence proofs.*

GCNs with Stale Features/Feature Gradients. Several recent works have been proposed to adopt either stale features (Chen et al., 2018; Cong et al., 2020) or feature gradients (Cong et al., 2021) in GCN training. Nevertheless, their convergence analysis considers only one of two kinds of staleness and derives a convergence rate of $\mathcal{O}(T^{-\frac{1}{2}})$ for pure sampling-based methods. This is, however, limited in distributed GCN training as its *convergence is simultaneously affected by both kinds of staleness*. PipeGCN proves such convergence with both stale features and feature gradients and offers a better rate of $\mathcal{O}(T^{-\frac{2}{3}})$. Furthermore, none of previous works has studied the errors incurred by staleness which harms the convergence speed, while PipeGCN develops a low-overhead smoothing method to reduce such errors.

3 The Proposed PipeGCN Framework

Overview. To enable efficient distributed GCN training, we first identify the two bottlenecks associated with vanilla partition-parallel training: *substantial communication overhead* and *frequently synchronized communication* with computation, and then address them directly by proposing a novel strategy, PipeGCN, to pipeline the communication and computation stages across two adjacent iterations in each partition of distributed GCN training, thus breaking the synchrony and hiding the communication overhead, as shown in Fig. 1 (c). It is non-trivial to achieve efficient GCN training with such a pipeline method, as staleness is incurred in communicated features/feature gradients and more importantly *little effort has been made to study the convergence guarantee of GCN training using stale feature gradients*. This work takes an initial effort to prove both the theoretical and



(a) Vanilla partition-parallel training of GCNs (per-partition view).

(b) **PipeGCN** (per-partition view).

Figure 2: A detailed comparison between vanilla partition-parallel training of GCNs and PipeGCN.

empirical convergence of such a pipeline GCN training method, and for the first time finds its convergence rate to be close to that of vanilla training without staleness. Furthermore, we propose a low-overhead smoothing method to reduce the errors due to stale features/feature gradients for further improving the convergence.

3.1 Bottlenecks in Vanilla Partition-Parallel Training

Significant communication overhead. Fig. 1 (a) illustrates *vanilla partition-parallel training*, where each partition holds *inner nodes* that come from the original graph and *boundary nodes* that come from other subgraphs. These *boundary nodes* are demanded by the *neighbor aggregate* of GCNs across neighbor partitions. E.g., in Fig. 1 (a), node-5 needs nodes-[3,4,6] residing on other partitions for calculating Equ. 1. Therefore, *it is the features/gradients of boundary nodes that dominate the communication overhead in distributed GCN training*. Note that the amount of *boundary nodes* can be excessive and far exceeds the inner nodes (up to $5.5\times$ in our evaluations), as the *boundary nodes* are *replicated* across partitions and scale with more partitions. Besides the sheer size, communication of *boundary nodes* occurs *for each layer* and *for both forward and backward passes*, making communication overhead substantial. We evaluate such overhead in Tab. 5 and find communication to be dominant, which is consistent with CAGNET (Tripathy et al., 2020).

Frequently synchronized communication. Note that this communication of *boundary nodes* must be finished before calculating Equ. 1 and Equ. 2, which *forces a synchronization between communication and computation and results in a full sequential execution*, as shown in Fig. 1 (b). Thus, for most of training time, each partition is waiting for tedious communication to finish before doing the actual compute work, and this repeats frequently for each GCN layer and for both forward and backward passes.

Table 2: The substantial communication overhead in vanilla partition-parallel training of GCNs. The *Comm. Ratio* is calculated by communication time divided by total training time. Detailed setting is in Sec. 4.

Dataset	# Partition	Comm. Ratio
Reddit	2	65.83%
	4	82.89%
ogbn-products	5	76.17%
	10	85.79%
Yelp	3	61.16%
	6	76.84%

3.2 The Proposed PipeGCN Method

Fig. 1 (c) illustrates the high-level overview of PipeGCN, which pipelines the *communicate* and *compute* stages spanning two iterations for *each GCN layer*. Fig. 2 further provides the detailed end-to-end flow, where PipeGCN removes the heavy communication overhead in the vanilla approach by breaking the synchronization between *communicate* and *compute* and hiding it with *compute* of each GCN layer. This is achieved by deferring the *communicate* to next iteration's *compute* (instead of serving the current iteration) such that *compute* and *communicate* can run in parallel. Inevitably, staleness is introduced in the deferred communication and results in a mixture usage of fresh inner features/gradients and staled boundary features/gradients.

Analytically, PipeGCN is achieved by modifying Equ. 1. For instance, when using a mean aggregator, Equ. 1 and its corresponding backward formulation become:

Algorithm 1: Training a GCN with PipeGCN (per-partition view).

```
Input: partition number $n$, partition id $i$, graph partition $\mathcal{G}_i$, propagation matrix $P_i$, node feature $X_i$, label
       $Y_i$, boundary node set $\mathcal{B}_i$, learning rate $\eta$, initial model $W_0$
Output: trained model $W_T$
$\mathcal{V}_i \leftarrow \{\text{node } v \in \mathcal{G}_i : v \notin \mathcal{B}_i\}$                    ▷ create inner node set
Broadcast $\mathcal{B}_i$ and Receive $[\mathcal{B}_1, \cdots, \mathcal{B}_n]$
$[\mathcal{S}_{i,1}, \cdots, \mathcal{S}_{i,n}] \leftarrow [\mathcal{B}_1 \cap \mathcal{V}_i, \cdots, \mathcal{B}_n \cap \mathcal{V}_i]$
Broadcast $\mathcal{V}_i$ and Receive $[\mathcal{V}_1, \cdots, \mathcal{V}_n]$
$[\mathcal{S}_{1,i}, \cdots, \mathcal{S}_{n,i}] \leftarrow [\mathcal{B}_i \cap \mathcal{V}_1, \cdots, \mathcal{B}_i \cap \mathcal{V}_n]$
$H^{(0)} \leftarrow \begin{bmatrix} X_i \\ 0 \end{bmatrix}$                    ▷ initialize node feature, set boundary feature as 0
for $t := 1 \rightarrow T$ do
    for $\ell := 1 \rightarrow L$ do                    ▷ forward pass
        if $t > 1$ then
            wait until $thread_f^{(\ell)}$ completes
            $[H^{(\ell-1)}_{\mathcal{S}_{1,i}}, \cdots, H^{(\ell-1)}_{\mathcal{S}_{n,i}}] \leftarrow [B^{(\ell)}_1, \cdots, B^{(\ell)}_n]$                    ▷ update boundary feature
        end
        with $thread_f^{(\ell)}$                    ▷ communicate boundary features in parallel
            Send $[H^{(\ell-1)}_{\mathcal{S}_{i,1}}, \cdots, H^{(\ell-1)}_{\mathcal{S}_{i,n}}]$ to partition $[1, \cdots, n]$ and Receive $[B^{(\ell)}_1, \cdots, B^{(\ell)}_n]$
        $H^{(\ell)}_{\mathcal{V}_i} \leftarrow \sigma(P_i H^{(\ell-1)} W^{(\ell)}_{t-1})$                    ▷ update inner nodes feature
    end
    $J^{(L)}_{\mathcal{V}_i} \leftarrow \frac{\mathrm{d}Loss(H^{(L)}_{\mathcal{V}_i}, Y_i)}{\mathrm{d}H^{(L)}_{\mathcal{V}_i}}$
    for $\ell := L \rightarrow 1$ do                    ▷ backward pass
        $G^{(\ell)}_i \leftarrow \left[P_i H^{(\ell-1)}\right]^\top \left(J^{(\ell)}_{\mathcal{V}_i} \circ \sigma'(P_i H^{(\ell-1)} W^{(\ell)}_{t-1})\right)$                    ▷ calculate weight gradient
        if $\ell > 1$ then
            $J^{(\ell-1)} \leftarrow P_i^\top \left(J^{(\ell)}_{\mathcal{V}_i} \circ \sigma'(P_i H^{(\ell-1)} W^{(\ell)}_{t-1})\right) [W^{(\ell)}_{t-1}]^\top$                    ▷ calculate feature gradient
            if $t > 1$ then
                wait until $thread_b^{(\ell)}$ completes
                for $j := 1 \rightarrow n$ do
                    $J^{(\ell-1)}_{\mathcal{S}_{i,j}} \leftarrow J^{(\ell-1)}_{\mathcal{S}_{i,j}} + C^{(\ell)}_j$                    ▷ accumulate feature gradient
                end
            end
            with $thread_b^{(\ell)}$                    ▷ communicate boundary feature gradient in parallel
                Send $[J^{(\ell-1)}_{\mathcal{S}_{1,i}}, \cdots, J^{(\ell-1)}_{\mathcal{S}_{n,i}}]$ to partition $[1, \cdots, n]$ and Receive $[C^{(\ell)}_1, \cdots, C^{(\ell)}_n]$
        end
    end
    $G \leftarrow AllReduce(G_i)$                    ▷ synchronize model gradient
    $W_t \leftarrow W_{t-1} - \eta G$                    ▷ update model
end
return $W_T$
```

$$z_v^{(t,\ell)} = \text{MEAN}\left(\{h_u^{(t,\ell-1)} \mid u \in \mathcal{N}(v) - \mathcal{B}(v)\} \cup \{h_u^{(t-1,\ell-1)} \mid u \in \mathcal{B}(v)\}\right) \tag{3}$$

$$\delta_{h_u}^{(t,\ell)} = \sum_{v: u \in \mathcal{N}(v) - \mathcal{B}(v)} \frac{1}{d_v} \cdot \delta_{z_v}^{(t,\ell+1)} + \sum_{v: u \in \mathcal{B}(v)} \frac{1}{d_v} \cdot \delta_{z_v}^{(t-1,\ell+1)} \tag{4}$$

where $\mathcal{B}$ is the set of boundary neighbors of v, d_v is the degree of node v, and $\delta_x^{(t,\ell)}$ is the gradient approximation of variable x at layer ℓ and iteration t. Lastly, PipeGCN's details are shown in Alg. 1.

3.3 PipeGCN's Convergence Guarantee

Staleness of communicated boundary features/gradients is the major challenge of PipeGCN, due to its unknown impact to the convergence. Here we provide convergence analysis under three assumptions:

Assumption 3.1. *The loss function $Loss(\cdot, \cdot)$ is C_{loss}-Lipschitz continuous and L_{loss}-smooth w.r.t. to the input node embedding vector, i.e., $|Loss(h^{(L)}, y) - Loss(h'^{(L)}, y)| \le C_{loss} \|h^{(L)} - h'^{(L)}\|_2$ and*

$\|\nabla Loss(h^{(L)}, y) - \nabla Loss(h'^{(L)}, y)\|_2 \le L_{loss}\|h^{(L)} - h'^{(L)}\|_2$ *where* h *is the predicted label and* y *is the correct label vector.*

Assumption 3.2. *The activation function* $\sigma(\cdot)$ *is* C_σ*-Lipschitz continuous and* L_σ*-smooth, i.e.,* $\|\sigma(z^{(\ell)}) - \sigma(z'^{(\ell)})\|_2 \le C_\sigma\|z^{(\ell)} - z'^{(\ell)}\|_2$ *and* $\|\sigma'(z^{(\ell)}) - \sigma'(z'^{(\ell)})\|_2 \le L_\sigma\|z^{(\ell)} - z'^{(\ell)}\|_2$.

Assumption 3.3. *For any* $\ell \in [L]$*, the norm of weight matrices, the propagation matrix, and the input feature matrix are bounded:* $\|W^{(\ell)}\|_F \le B_W, \|P\|_F \le B_P, \|X\|_F \le B_X$*. (This generic assumption is also used in (Chen et al., 2018; Liao et al., 2020; Garg et al., 2020; Cong et al., 2021).)*

Then we provide the convergence rate of PipeGCN in the following theorem:

Theorem 3.1. *Under Assumptions 3.1, 3.2, and 3.3, we can derive the following by choosing a learning rate* $\eta = \frac{\sqrt{\varepsilon}}{E}$ *and number of training iterations* $T = (\mathcal{L}(\theta^{(1)}) - \mathcal{L}(\theta^*))E\varepsilon^{-\frac{3}{2}}$*:*

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla\mathcal{L}(\theta^{(t)})\|_2 \le \mathcal{O}(\varepsilon)$$

where $\varepsilon > 0$ *is an arbitrarily small constant,* $\mathcal{L}(\cdot)$ *is the loss function,* $\theta^{(t)}$ *is the parameter vector at iteration* t*, and*

$$U = B_P B_W C_\sigma, U_{\max} = \max\{1, U^L\}, S = U_{\max}^6 L^2 B_X^2 B_P^2 C_\sigma C_{loss} U$$

$$R = C_{loss}\left(2L_\sigma S + U_{\max}^3 C_\sigma^2 C_{loss} B_X L\right)$$

$$E = LB_P(U_{\max}B_X(C_\sigma^2 S(C_{loss} + L_{loss}) + R(LU_{\max} + U^L)) + C_\sigma^2 C_{loss} S)$$

Therefore ***the convergence rate of PipeGCN is*** $\mathcal{O}(T^{-\frac{2}{3}})$***, which is better than sampling-based method*** ($\mathcal{O}(T^{-\frac{1}{2}})$) (Chen et al., 2018; Cong et al., 2021) ***and close to full-graph training*** ($\mathcal{O}(T^{-1})$). The detailed proof can be found in Appendix A.

3.4 The Proposed Smoothing Method

The proposed smoothing method aims at reducing errors incurred by stale features/(feature gradients) at a minimal overhead. Here we take the smoothing of feature gradient as an example, but the same formulation also applies to stale features. To improve the approximate gradients for each feature, fluctuations in feature gradients between adjacent iterations should be reduced. Therefore, we apply a light-weight moving average for the feature gradients of each boundary node v as follow:

$$\hat{\delta}_{z_v}^{(t,\ell)} = \gamma\hat{\delta}_{z_v}^{(t-1,\ell)} + (1-\gamma)\delta_{z_v}^{(t-1,\ell)}$$

where $\hat{\delta}_{z_v}^{(t,\ell)}$ is the smoothed feature gradient at layer ℓ and iteration t, and γ is the decay rate. To integrate the smoothed feature gradient into the backward pass, we rewrite Equ. 4 as:

$$\hat{\delta}_{h_u}^{(t,\ell)} = \sum_{v:u\in\mathcal{N}(v)-\mathcal{B}(v)}\frac{1}{d_v}\cdot\delta_{z_v}^{(t,\ell+1)} + \sum_{v:u\in\mathcal{B}(v)}\frac{1}{d_v}\cdot\hat{\delta}_{z_v}^{(t-1,\ell+1)}$$

Such smoothing of stale features and gradients can thus be independently applied to PipeGCN.

4 Experiment Results

We evaluate PipeGCN on four large-scale datasets, Reddit (Hamilton et al., 2017), ogbn-products (Hu et al., 2020), Yelp (Zeng et al., 2020), and ogbn-papers100M (Hu et al., 2020). More details are provided in Tab. 3. *To ensure robustness and reproducibility, we fix (i.e., do not tune) the hyper-parameters and settings for PipeGCN and its variants throughout all experiments.* To implement partition parallelism (for both vanilla distributed GCN training and PipeGCN), the widely used METIS (Karypis & Kumar, 1998) partition algorithm is adopted for graph partition with its objective set to minimize the communication volume. We implement PipeGCN in PyTorch (Paszke et al., 2019) and DGL (Wang et al., 2019). Experiments are conducted on a machine with 10 RTX-2080Ti (11GB), Xeon 6230R@2.10GHz (187GB), and PCIe3x16 connecting CPU-GPU and GPU-GPU. Only for ogbn-papers100M, we use 4 computational nodes (each contains 8 MI60 GPUs, an AMD EPYC 7642 CPU, and 48 lane PCI 3.0 connecting CPU-GPU and GPU-GPU) networked with 10Gbps Ethernet. To support full-graph GCN training with the model sizes in Tab. 3, the minimum required partition numbers are 2, 3, 5, 4 for Reddit, ogbn-products, Yelp, and ogbn-papers100M, respectively.

Table 3: Detailed experiment setups: graph datasets, GCN models, and training hyper-parameters.

Dataset	# Nodes	# Edges	Feat. size	GraphSAGE model size	Optimizer	LearnRate	Dropout	# Epoch
Reddit	233K	114M	602	4 layer, 256 hidden units	Adam	0.01	0.5	3000
ogbn-products	2.4M	62M	100	3 layer, 128 hidden units	Adam	0.003	0.3	500
Yelp	716K	7.0M	300	4 layer, 512 hidden units	Adam	0.001	0.1	3000
ogbn-papers100M	111M	1.6B	128	3 layer, 48 hidden units	Adam	0.01	0.5	1000

For convenience, we here name all methods: vanilla partition-parallel training of GCNs (**GCN**), PipeGCN with feature gradient smoothing (**PipeGCN-G**), PipeGCN with feature smoothing (**PipeGCN-F**), and PipeGCN with both smoothing (**PipeGCN-GF**). *The default decay rate γ for all smoothing methods is set to 0.95.*

4.1 Improving Training Throughput over Full-Graph Training Methods

Fig. 3 compares the training throughput between PipeGCN and SOTA full-graph training methods (ROC (Jia et al., 2020) and CAGNET (Tripathy et al., 2020)). We observe that both vanilla partition-parallel training (GCN) and PipeGCN greatly outperform ROC and CATNET across different number of partitions, because they avoid both the expensive CPU-GPU swaps (ROC) and the redundant node broadcast (CAGNET). Specifically, GCN is **3.1**$\sim$**6.2**× faster than ROC and **2.1**$\sim$**7.9**× faster than CAGNET (c=2). PipeGCN further improves upon GCN, achieving a throughput improvement of **5.6**$\sim$**9.9**× over ROC and **3.9**$\sim$**14.7**× over CAGNET (c=2). The comparison on more datasets can be found in the Appendix B, which consistently show the advantages of PipeGCN. Furthermore, we further provide the epoch time breakdown of ROC and CAGNET on Reddit in the Appendix F for understanding where PipeGCN gains significant savings over the baseline algorithms. Considering the substantial performance gap between ROC/CAGNET and GCN, we focus on comparing GCN with PipeGCN for the reminder of the section. Note that we are not able to compare with NeuGraph (Ma et al., 2019), AliGraph (Zhu et al., 2019), and P^3 (Gandhi & Iyer, 2021) as their codes are not open source.

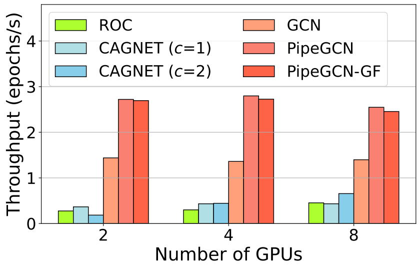


Figure 3: Throughput comparison on Reddit. Each partition uses one GPU (except CAGNET (c=2) uses two).

4.2 Improving Training Throughput without Compromising Accuracy

We compare the training performance of both test score and training throughput between GCN and PipeGCN in Tab. 4. We can see that *PipeGCN without smoothing already achieves a comparable test score with the vanilla GCN training* on both Reddit and Yelp, and incurrs only a negligible accuracy drop (-0.08%$\sim$-0.23%) on ogbn-products, while boosting the training throughput by **1.72**× $\sim$ **2.16**× across all datasets and different number of partitions, thus validating the effectiveness of PipeGCN.

With the proposed smoothing method plugged in, *PipeGCN-G/F/GF is able to compensate the dropped score of vanilla PipeGCN, achieving an equal or even better test score as/than the vanilla GCN training* (without staleness), e.g., 97.14% vs. 97.11% on Reddit, 79.36% vs. 79.14% on ogbn-products and 65.28% vs. 65.26% on Yelp. Meanwhile, PipeGCN-G/F/GF enjoys a similar throughput improvement as vanilla PipeGCN, thus validating the negligible overhead of the proposed smoothing method. Therefore, ***pipelined transfer of features and gradients greatly improves the training throughput while maintaining the full-graph accuracy***.

Note that our distributed GCN training methods consistently achieve higher test scores than SOTA sampling-based methods for GraphSAGE-based models reported in (Zeng et al., 2020) and (Hu et al., 2020), confirming that the full-graph training technique is preferred to obtain better GCN models. For example, the best sampling-based method achieves a 96.6% accuracy on Reddit (Zeng et al., 2020) while full-graph GCN training achieves 97.1%, and PipeGCN improves the accuracy by 0.28% over sampling-based GraphSAGE models on ogbn-products (Hu et al., 2020). Such an advantage of full-graph training is also validated by recent works (Jia et al., 2020; Tripathy et al., 2020).

4.3 Maintaining Convergence Speed

To understand PipeGCN's influence on the convergence speed, we compare the training curve among different methods in Fig. 4. We observe that the convergence of PipeGCN without smoothing is still comparable with that of the vanilla GCN training, although PipeGCN converges slower at the

Table 4: Training performance comparison among vanilla partition-parallel training (GCN) and PipeGCN variants (PipeGCN*), where we report the test accuracy for Reddit and ogbn-products, and the F1-micro score for Yelp. Highest performance is in bold.

Dataset	Method	Test Score (%)	Throughput
Reddit (2 partitions)	GCN	97.11±0.02	1× (1.94 epochs/s)
	PipeGCN	97.12±0.02	**1.91**×
	PipeGCN-G	**97.14±0.03**	1.89×
	PipeGCN-F	97.09±0.02	1.89×
	PipeGCN-GF	97.12±0.02	1.87×
Reddit (4 partitions)	GCN	**97.11±0.02**	1× (2.07 epochs/s)
	PipeGCN	97.04±0.03	**2.12**×
	PipeGCN-G	97.09±0.03	2.07×
	PipeGCN-F	97.10±0.02	2.10×
	PipeGCN-GF	97.10±0.02	2.06×
ogbn-products (5 partitions)	GCN	79.14±0.35	1× (1.45 epochs/s)
	PipeGCN	79.06±0.42	**1.94**×
	PipeGCN-G	79.20±0.38	1.90×
	PipeGCN-F	**79.36±0.38**	1.90×
	PipeGCN-GF	78.86±0.34	1.91×
ogbn-products (10 partitions)	GCN	79.14±0.35	1× (1.28 epochs/s)
	PipeGCN	78.91±0.65	**1.87**×
	PipeGCN-G	79.08±0.58	1.82×
	PipeGCN-F	**79.21±0.31**	1.81×
	PipeGCN-GF	78.77±0.23	1.82×
Yelp (3 partitions)	GCN	65.26±0.02	1× (2.00 epochs/s)
	PipeGCN	**65.27±0.01**	**2.16**×
	PipeGCN-G	65.26±0.02	2.15×
	PipeGCN-F	65.26±0.03	2.15×
	PipeGCN-GF	65.26±0.04	2.11×
Yelp (6 partitions)	GCN	65.26±0.02	1× (2.25 epochs/s)
	PipeGCN	65.24±0.02	**1.72**×
	PipeGCN-G	**65.28±0.02**	1.69×
	PipeGCN-F	65.25±0.04	1.68×
	PipeGCN-GF	65.26±0.04	1.67×

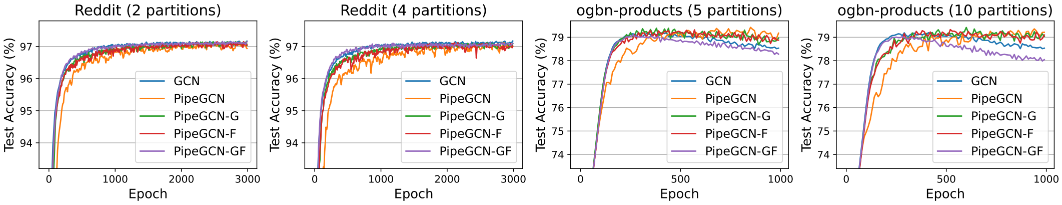


Figure 4: Epoch-to-accuracy comparison among vanilla partition-parallel training (GCN) and PipeGCN variants (PipeGCN*), where *PipeGCN and its variants achieve a similar convergence as the vanilla training (without staleness) but are twice as fast in wall-clock time* (see Tab. 4).

early phase of training and then catches up at the later phase, due to the staleness of boundary features/gradients. With the proposed smoothing methods, *PipeGCN-G/F boosts the convergence substantially and matches or even outperforms (esp. at late training phase) the convergence speed of vanilla GCN training*. There is no clear difference between PipeGCN-G and PipeGCN-F. Lastly, with combined smoothing of features and gradients, *PipeGCN-GF can acheive the same convergence speed as vanilla GCN training* (e.g., on Reddit) but can overfit gradually similar to the vanilla GCN training, which is further investigated in Sec. 4.4. Therefore, ***PipeGCN maintains the convergence speed w.r.t the number of epochs while reduces the end-to-end training time by around 50% thanks to its boosted training throughput*** (see Tab. 4).

4.4 Benefit of PipeGCN with Staleness Smoothing

Error Reduction and Convergence Speedup. To understand why the proposed smoothing technique (see Sec. 3.4) speeds up convergence, we compare the error incurred by the stale communication between PipeGCN and PipeGCN-G/F. The error is calculated as the Frobenius-norm of the gap between the correct gradient/feature and the stale gradient/feature used in PipeGCN training. Fig. 5

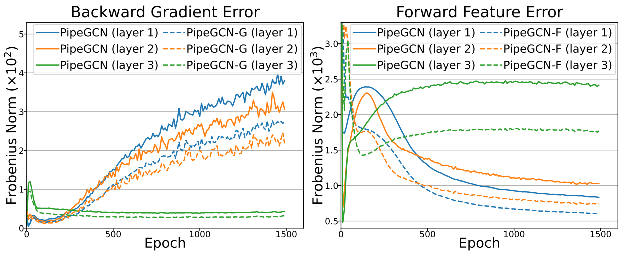


Figure 5: Comparison of the resulting feature gradient error and feature error from PipeGCN and PipeGCN-G/F at each GCN layer on Reddit (2 partitions). PipeGCN-G/F here uses a default smoothing decay rate of 0.95.

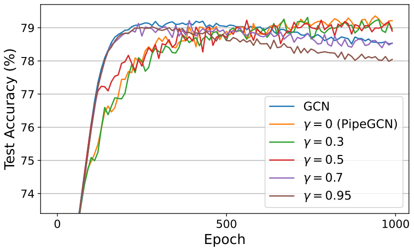


Figure 6: Test-accuracy convergence comparison among different smoothing decay rates γ in PipeGCN-GF on ogbn-products (10 partitions).

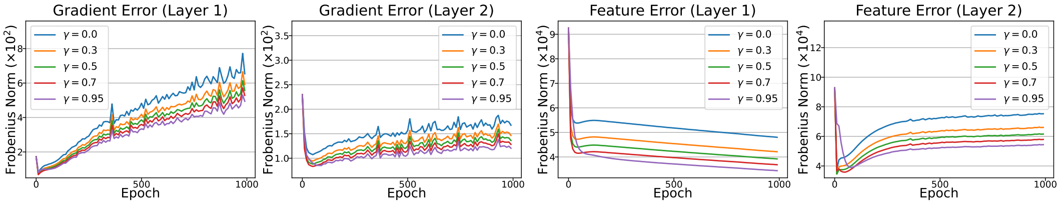


Figure 7: Comparison of the resulting feature gradient error and feature error when adopting different decay rates γ at each GCN layer on ogbn-products (10 partitions).

compares the error at each GCN layer. We can see that *the proposed smoothing technique (PipeGCN-G/F) reduces the error of staleness substantially* (from the base version of PipeGCN) and this benefit consistently holds across different layers in terms of both feature and gradients errors, validating the effectiveness of our smoothing method and explains its improvement to the convergence speed.

Overfitting Reduction. To understand the effect of staleness smoothing on model overfitting, we also evaluate the test-accuracy convergence under different decay rates γ in Fig. 6. Here ogbn-products is adopted as the study case because the distribution of its test set largely differs from that of its training set. From Fig. 6, we observe that smoothing with a large γ (0.7/0.95) offers a fast convergence, i.e., close to the vanilla GCN training, but overfits rapidly. To understand this issue, we further provide detailed comparisons of the errors incurred under different γ in Fig. 7. We can see that a larger γ enjoys lower approximation errors and makes the gradients/features more stable, thus improving the convergence speed. The increased stability on the training set, however, constrains the model from exploring a more general minimum point on the test set, thus leading to overfitting as the vanilla GCN training. In contrast, *a small* γ *(*$0 \sim 0.5$*) mitigates this overfitting and achieves a better accuracy* (see Fig. 6). But a too-small γ (e.g., 0) gives a high error for both stale features and gradients (see Fig. 7), thus suffering from a slower convergence. Therefore, a trade-off between convergence speed and achievable optimality exists between different smoothing decay rates, and $\gamma = 0.5$ combines the best of both worlds in this study.

4.5 Training Time Improvement Breakdown

To further understand the training time improvement of PipeGCN, we breakdown the epoch time into three parts (intra-partition computation, inter-partition communication, and reduce of model gradient) and provide an example in Tab. 5. More results can be found in Appendix D.

Table 5: Training time breakdown on ogbn-papers100M.

Method	Total	Comm.	Reduce
GCN	10.5s	6.6s	1.2s
PipeGCN	6.5s	2.6s	1.2s
PipeGCN-GF	6.7s	2.8s	1.1s

5 Conclusion

In this work, we propose a new method, PipeGCN, for efficient full-graph GCN training. PipeGCN pipelines communication with computation in distributed GCN training to hide the substantial communication overhead. Furthermore, we take an initial effort to understand the convergence of GCN training with both stale features and feature gradients, and further propose a smoothing method to speedup the convergence of vanilla PipeGCN. Extensive experiments validate the advantages of PipeGCN over both vanilla training (without staleness) and SOTA full-graph training.

REFERENCES

Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pp. 942–950. PMLR, 2018.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui 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*, pp. 257–266, 2019.

Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1393–1403, 2020.

Weilin Cong, Morteza Ramezani, and Mehrdad Mahdavi. On the importance of sampling in learning graph convolutional networks. *arXiv preprint arXiv:2103.02696*, 2021.

Swapnil Gandhi and Anand Padmanabha Iyer. P3: Distributed deep graph learning at scale. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pp. 551–568, 2021.

Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *International Conference on Machine Learning*, pp. 3419–3430. PMLR, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

Qirong Ho, James Cipar, Henggang Cui, Jin Kyu Kim, Seunghak Lee, Phillip B Gibbons, Garth A Gibson, Gregory R Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. *Advances in neural information processing systems*, 2013:1223, 2013.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with roc. *Proceedings of Machine Learning and Systems (MLSys)*, pp. 187–198, 2020.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27:19–27, 2014.

Youjie Li, Mingchao Yu, Songze Li, Salman Avestimehr, Nam Sung Kim, and Alexander Schwing. Pipe-sgd: A decentralized pipelined sgd framework for distributed deep net training. *arXiv preprint arXiv:1811.03619*, 2018.

Renjie Liao, Raquel Urtasun, and Richard Zemel. A pac-bayesian approach to generalization bounds for graph neural networks. *arXiv preprint arXiv:2012.07690*, 2020.

Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. Neugraph: parallel deep neural network computation on large graphs. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pp. 443–458, 2019.

Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730*, 2011.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019.

John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, et al. Dorylus: affordable, scalable, and accurate gnn training with distributed cpu servers and serverless threads. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pp. 495–514, 2021.

Alok Tripathy, Katherine Yelick, and Aydin Buluc. Reducing communication in graph neural network training. *arXiv preprint arXiv:2005.03300*, 2020.

Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Bowen Yang, Jian Zhang, Jonathan Li, Christopher Ré, Christopher Aberger, and Christopher De Sa. Pipemare: Asynchronous pipeline parallel dnn training. *Proceedings of Machine Learning and Systems*, 3, 2021.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2020.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 5165–5175, 2018.

Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. Aligraph: A comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730*, 2019.

A Convergence Proof

In this section, we prove the convergence of PipeGCN. The essential step is to prove that the bound of the gradient error is controlled by learning rate η.

A.1 Notations

For a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an adjacency matrix A, feature matrix X, we define the propagation matrix P as $P \coloneqq \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$, where $\tilde{A} = A + I, \tilde{D}_{u,u} = \sum_v \tilde{A}_{u,v}$. One GCN layer performs one step of feature propagation (Kipf & Welling, 2016) as formulated below

$$\begin{aligned} H^{(0)} &= X \\ Z^{(\ell)} &= PH^{(\ell-1)}W^{(\ell)} \\ H^{(\ell)} &= \sigma(Z^{(\ell)}) \end{aligned}$$

where $H^{(\ell)}$, $W^{(\ell)}$, and $Z^{(\ell)}$ denote the embedding matrix, the trainable weight matrix, and the intermediate embedding matrix in the ℓ-th layer, respectively, and σ denotes the non-linear function. For an L-layer GCN, the loss function is denoted by $\mathcal{L}(\theta)$ where $\theta = \text{vec}[W^{(1)}, W^{(2)}, \cdots, W^{(L)}]$. We define the ℓ-th layer as a function $f^{(\ell)}(\cdot, \cdot)$.

$$f^{(\ell)}(H^{(\ell-1)}, W^{(\ell)}) \coloneqq \sigma(PH^{(\ell-1)}W^{(\ell)})$$

and its gradient w.r.t. the input embedding matrix can be represented as

$$J^{(\ell-1)} = \nabla_H f^{(\ell)}(J^{(\ell)}, H^{(\ell-1)}, W^{(\ell)}) \coloneqq P^\top M^{(\ell)}[W^{(\ell)}]^\top$$

and its gradient w.r.t. the weight can be represented as

$$G^{(\ell)} = \nabla_W f^{(\ell)}(J^{(\ell)}, H^{(\ell-1)}, W^{(\ell)}) \coloneqq [PH^{(\ell-1)}]^\top M^{(\ell)}$$

where $M^{(\ell)} = J^{(\ell)} \circ \sigma'(Z^{(\ell)})$ and $\circ$ denotes Hadamard product.

For partition-parallel training, we can split P into two parts $P = P_{in} + P_{bd}$ where P_{in} represents inter-partition propagation and P_{bd} denotes intra-partition propagation. For PipeGCN, we can represent one GCN layer as below

$$\begin{aligned} \widetilde{H}^{(t,0)} &= X \\ \widetilde{Z}^{(t,\ell)} &= P_{in}\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t,\ell)} \\ \widetilde{H}^{(t,\ell)} &= \sigma(\widetilde{Z}^{(t,\ell)}) \end{aligned}$$

where t is the epoch number and $\widetilde{W}^{(t,\ell)}$ is the weight at epoch t layer ℓ. We define the loss function for this setting as $\widetilde{\mathcal{L}}(\widetilde{\theta}^{(t)})$ where $\widetilde{\theta}^{(t)} = \text{vec}[\widetilde{W}^{(t,1)}, \widetilde{W}^{(t,2)}, \cdots, \widetilde{W}^{(t,L)}]$. We can also summarize the layer as a function $\widetilde{f}^{(t,\ell)}(\cdot, \cdot)$

$$\widetilde{f}^{(t,\ell)}(\widetilde{H}^{(t,\ell-1)}, \widetilde{W}^{(t,\ell)}) \coloneqq \sigma(P_{in}\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t,\ell)})$$

Note that $\widetilde{H}^{(t-1,\ell-1)}$ is not a part of the input of $\widetilde{f}^{(t,\ell)}(\cdot, \cdot)$ because it is a constant for the t-th epoch. The corresponding backward propagation follows the following computation

$$\begin{aligned} \widetilde{J}^{(t,\ell-1)} &= \nabla_H \widetilde{f}^{(t,\ell)}(\widetilde{J}^{(t,\ell)}, \widetilde{H}^{(t,\ell-1)}, \widetilde{W}^{(t,\ell)}) \\ \widetilde{G}^{(t,\ell)} &= \nabla_W \widetilde{f}^{(t,\ell)}(\widetilde{J}^{(t,\ell)}, \widetilde{H}^{(t,\ell-1)}, \widetilde{W}^{(t,\ell)}) \end{aligned}$$

where

$$\begin{aligned} \widetilde{M}^{(t,\ell)} &= \widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t,\ell)}) \\ \nabla_H \widetilde{f}^{(t,\ell)}(\widetilde{J}^{(t,\ell)}, \widetilde{H}^{(t,\ell-1)}, \widetilde{W}^{(t,\ell)}) &\coloneqq P_{in}^\top \widetilde{M}^{(t,\ell)}[\widetilde{W}^{(t,\ell)}]^\top + P_{bd}^\top \widetilde{M}^{(t-1,\ell)}[\widetilde{W}^{(t-1,\ell)}]^\top \\ \nabla_W \widetilde{f}^{(t,\ell)}(\widetilde{J}^{(t,\ell)}, \widetilde{H}^{(t,\ell-1)}, \widetilde{W}^{(t,\ell)}) &\coloneqq [P_{in}\widetilde{H}^{(t,\ell-1)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)}]^\top \widetilde{M}^{(t,\ell)} \end{aligned}$$

Again, $\widetilde{J}^{(t-1,\ell)}$ is not a part of the input of $\nabla_H \widetilde{f}^{(t,\ell)}(\cdot, \cdot, \cdot)$ or $\nabla_W \widetilde{f}^{(t,\ell)}(\cdot, \cdot, \cdot)$ because it is a constant for epoch t. Finally, we define $\nabla\widetilde{\mathcal{L}}(\widetilde{\theta}^{(t)}) = \text{vec}[\widetilde{G}^{(t,1)}, \widetilde{G}^{(t,2)}, \cdots, \widetilde{G}^{(t,L)}]$. It should be highlighted that the 'gradient' $\nabla_H \widetilde{f}^{(t,\ell)}(\cdot, \cdot, \cdot)$, $\nabla_W \widetilde{f}^{(t,\ell)}(\cdot, \cdot, \cdot)$ and $\nabla\widetilde{\mathcal{L}}(\widetilde{\theta}^{(t)})$ are not the correct gradient for the corresponding forward process due to the stale communication. Properties of gradient cannot be directly applied to these variables.

The definitions of other constants $C_{\text{loss}}, L_{\text{loss}}, C_\sigma, L_\sigma, B_W, B_P, B_X$ can be found in Assumption 3.1-3.3 of the main content.

A.2 Bounded Matrices and Changes

Lemma A.1. *For any $\ell \in [L]$, the Frobenius norm of node embedding matrices, gradient passing from the ℓ-th layer node embeddings to the $(\ell - 1)$-th, gradient matrices are bounded, i.e.,*

$$\|H^{(\ell)}\|_F, \|\widetilde{H}^{(t,\ell)}\|_F \leq B_H,$$

$$\|J^{(\ell)}\|_F, \|\widetilde{J}^{(t,\ell)}\|_F \leq B_J,$$

$$\|M^{(\ell)}\|_F, \|\widetilde{M}^{(t,\ell)}\|_F \leq B_M,$$

$$\|G^{(\ell)}\|_F, \|\widetilde{G}^{(t,\ell)}\|_F \leq B_G$$

where

$$B_H = \max_\ell (C_\sigma B_P B_W)^\ell B_X$$

$$B_J = \max_\ell (C_\sigma B_P B_W)^\ell C_{loss}$$

$$B_M = C_\sigma B_J$$

$$B_G = B_P B_H B_M$$

Proof. The proof of $\|H^{(\ell)}\|_F \leq B_H$ and $\|J^{(\ell)}\|_F \leq B_J$ can be found in Proposition 2 in (Cong et al., 2021). By induction,

$$\begin{aligned}
\|\widetilde{H}^{(t,\ell)}\|_F &= \|\sigma(P_{in}\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t,\ell)})\|_F \\
&\leq C_\sigma B_W \|P_{in} + P_{bd}\|_F (C_\sigma B_P B_W)^{\ell-1} B_X \\
&\leq (C_\sigma B_P B_W)^\ell B_X
\end{aligned}$$

$$\begin{aligned}
\|\widetilde{J}^{(t,\ell-1)}\|_F &= \left\| P_{in}^\top \left(\widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t,\ell)})\right) [\widetilde{W}^{(t,\ell)}]^\top + P_{bd}^\top \left(\widetilde{J}^{(t-1,\ell)} \circ \sigma'(\widetilde{Z}^{(t-1,\ell)})\right) [\widetilde{W}^{(t-1,\ell)}]^\top \right\|_F \\
&\leq C_\sigma B_W \|P_{in} + P_{bd}\|_F (C_\sigma B_P B_W)^{L-\ell} C_{\text{loss}} \\
&\leq (C_\sigma B_P B_W)^{L-\ell+1} C_{\text{loss}}
\end{aligned}$$

$$\|M^{(\ell)}\|_F = \|J^{(\ell)} \circ \sigma'(Z^{(\ell)})\|_F \leq C_\sigma B_J$$

$$\|\widetilde{M}^{(t,\ell)}\|_F = \|\widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t,\ell)})\|_F \leq C_\sigma B_J$$

$$\begin{aligned}
G^{(\ell)} &= [PH^{(\ell-1)}]^\top M^{(\ell)} \\
&\leq B_P B_H B_M
\end{aligned}$$

$$\begin{aligned}
\widetilde{G}^{(t,\ell)} &= [P_{in}\widetilde{H}^{(t,\ell-1)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)}]^\top \widetilde{M}^{(t,\ell)} \\
&\leq B_P B_H B_M
\end{aligned}$$

□

Because the gradient matrices are bounded, the weight change is bounded.

Corollary A.1. *For any t, ℓ, $\|\widetilde{W}^{(t,\ell)} - \widetilde{W}^{(t-1,\ell)}\|_F \leq B_{\Delta W} = \eta B_G$ where η is the learning rate.*

Now we can analyze the changes of intermediate variables.

Lemma A.2. *For any t, ℓ, we have $\|\widetilde{Z}^{(t,\ell)} - \widetilde{Z}^{(t-1,\ell)}\|_F \leq B_{\Delta Z}$, $\|\widetilde{H}^{(t,\ell)} - \widetilde{H}^{(t-1,\ell)}\|_F \leq B_{\Delta H}$, where $B_{\Delta Z} = \sum_{i=0}^{L-1} C_\sigma^i B_P^{i+1} B_W^i B_H B_{\Delta W}$ and $B_{\Delta H} = C_\sigma B_{\Delta Z}$.*

Proof. When $\ell = 0$, $\|\widetilde{H}^{(t,0)} - \widetilde{H}^{(t-1,0)}\|_F = \|X - X\|_F = 0$. Now we consider $\ell > 0$ by induction.

$$\begin{aligned}\|\widetilde{Z}^{(t,\ell)} - \widetilde{Z}^{(t-1,\ell)}\|_F =& \|(P_{in}\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t,\ell)}) \\ &- (P_{in}\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t-1,\ell)} + P_{bd}\widetilde{H}^{(t-2,\ell-1)}\widetilde{W}^{(t-1,\ell)})\|_F \\ =& \|P_{in}(\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} - \widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t-1,\ell)}) \\ &+ P_{bd}(\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t,\ell)} - \widetilde{H}^{(t-2,\ell-1)}\widetilde{W}^{(t-1,\ell)})\|_F\end{aligned}$$

Then we analyze the bound of $s^{(t,\ell)} := \|\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} - \widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t-1,\ell)}\|_F$.

$$\begin{aligned}s^{(t,\ell)} &\leq \|\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} - \widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t-1,\ell)}\|_F + \|\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t-1,\ell)} - \widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t-1,\ell)}\|_F \\ &\leq B_H\|\widetilde{W}^{(t,\ell)} - \widetilde{W}^{(t-1,\ell)}\|_F + B_W\|\widetilde{H}^{(t,\ell-1)} - \widetilde{H}^{(t-1,\ell-1)}\|_F\end{aligned}$$

According to A.1, $\|\widetilde{W}^{(t,\ell)} - \widetilde{W}^{(t-1,\ell)}\|_F \leq B_{\Delta W}$. By induction, $\|\widetilde{H}^{(t,\ell-1)} - \widetilde{H}^{(t-1,\ell-1)}\|_F \leq \sum_{i=0}^{\ell-2} C_\sigma^{i+1} B_P^{i+1} B_W^i B_H B_{\Delta W}$. Combining these inequalities,

$$s^{(t,\ell)} \leq B_H B_{\Delta W} + \sum_{i=1}^{\ell-1} C_\sigma^i B_P^i B_W^i B_H B_{\Delta W}$$

Plugging it back, we have

$$\begin{aligned}\|\widetilde{Z}^{(t,\ell)} - \widetilde{Z}^{(t-1,\ell)}\|_F \leq& \|P_{in}(\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} - \widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t-1,\ell)}) \\ &+ P_{bd}(\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t,\ell)} - \widetilde{H}^{(t-2,\ell-1)}\widetilde{W}^{(t-1,\ell)})\|_F \\ \leq& B_P\left(B_H B_{\Delta W} + \sum_{i=1}^{\ell-1} C_\sigma^i B_P^i B_W^i B_H B_{\Delta W}\right) \\ =& \sum_{i=0}^{\ell-1} C_\sigma^i B_P^{i+1} B_W^i B_H B_{\Delta W}\end{aligned}$$

$$\begin{aligned}\|\widetilde{H}^{(t,\ell)} - \widetilde{H}^{(t-1,\ell)}\|_F =& \|\sigma(\widetilde{Z}^{(t,\ell)}) - \sigma(\widetilde{Z}^{(t-1,\ell)})\|_F \\ \leq& C_\sigma\|\widetilde{Z}^{(t,\ell)} - \widetilde{Z}^{(t-1,\ell)}\|_F \\ \leq& C_\sigma B_{\Delta Z}\end{aligned}$$

□

Lemma A.3. $\|\widetilde{J}^{(t,\ell)} - \widetilde{J}^{(t-1,\ell)}\|_F \leq B_{\Delta J}$ *where*

$$B_{\Delta J} = \max_\ell (B_P B_W C_\sigma)^\ell B_{\Delta H} L_{loss} + (B_M B_{\Delta W} + L_\sigma B_J B_{\Delta Z} B_W) \sum_{i=0}^{L-1} B_P^{i+1} B_W^i C_\sigma^i$$

Proof. For the last layer ($\ell = L$), $\|\widetilde{J}^{(t,L)} - \widetilde{J}^{(t-1,L)}\|_F \leq L_{\text{loss}}\|\widetilde{H}^{(t,L)} - \widetilde{H}^{(t-1,L)}\|_F \leq L_{\text{loss}} B_{\Delta H}$. For the case of $\ell < L$, we prove the lemma by using induction.

$$\begin{aligned}\|\widetilde{J}^{(t,\ell-1)} - \widetilde{J}^{(t-1,\ell-1)}\|_F =& \left\| \left(P_{in}^\top \widetilde{M}^{(t,\ell)}[\widetilde{W}^{(t,\ell)}]^\top + P_{bd}^\top \widetilde{M}^{(t-1,\ell)}[\widetilde{W}^{(t-1,\ell)}]^\top\right)\right. \\ &\left. - \left(P_{in}^\top \widetilde{M}^{(t-1,\ell)}[\widetilde{W}^{(t-1,\ell)}]^\top + P_{bd}^\top \widetilde{M}^{(t-2,\ell)}[\widetilde{W}^{(t-2,\ell)}]^\top\right) \right\|_F \\ \leq& \left\| P_{in}^\top \left(\widetilde{M}^{(t,\ell)}[\widetilde{W}^{(t,\ell)}]^\top - \widetilde{M}^{(t-1,\ell)}[\widetilde{W}^{(t-1,\ell)}]^\top\right)\right\|_F \\ &+ \left\| P_{bd}^\top \left(\widetilde{M}^{(t-1,\ell)}[\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t-2,\ell)}[\widetilde{W}^{(t-2,\ell)}]^\top\right)\right\|_F\end{aligned}$$

We define $s^{(t,\ell)} := \left\| \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top - \widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top \right\|_F$ and analyze its bound.

$$\begin{aligned}
s^{(t,\ell)} \leq & \left\| \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top - \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top \right\|_F \\
& + \left\| \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top \right\|_F \\
\leq & B_M \left\| [\widetilde{W}^{(t,\ell)}]^\top - [\widetilde{W}^{(t-1,\ell)}]^\top \right\|_F + B_W \left\| \widetilde{M}^{(t,\ell)} - \widetilde{M}^{(t-1,\ell)} \right\|_F
\end{aligned}$$

According to Corollary A.1, $\left\| [\widetilde{W}^{(t,\ell)}]^\top - [\widetilde{W}^{(t-1,\ell)}]^\top \right\|_F \leq B_{\Delta W}$. For the second term,

$$\begin{aligned}
& \|\widetilde{M}^{(t,\ell)} - \widetilde{M}^{(t-1,\ell)}\|_F \\
= & \|\widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t,\ell)}) - \widetilde{J}^{(t-1,\ell)} \circ \sigma'(\widetilde{Z}^{(t-1,\ell)})\|_F \\
\leq & \|\widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t,\ell)}) - \widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t-1,\ell)})\|_F + \|\widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t-1,\ell)}) - \widetilde{J}^{(t-1,\ell)} \circ \sigma'(\widetilde{Z}^{(t-1,\ell)})\|_F \\
\leq & B_J \|\sigma'(\widetilde{Z}^{(t,\ell)}) - \sigma'(\widetilde{Z}^{(t-1,\ell)})\|_F + C_\sigma \|\widetilde{J}^{(t,\ell)} - \widetilde{J}^{(t-1,\ell)}\|_F \qquad (5)
\end{aligned}$$

According to the smoothness of σ and Lemma A.2, $\|\sigma'(\widetilde{Z}^{(t,\ell)}) - \sigma'(\widetilde{Z}^{(t-1,\ell)})\|_F \leq L_\sigma B_{\Delta Z}$. By induction,

$$\begin{aligned}
& \|\widetilde{J}^{(t,\ell)} - \widetilde{J}^{(t-1,\ell)}\|_F \\
& \leq (B_P B_W C_\sigma)^{(L-\ell)} B_{\Delta H} L_{\text{loss}} + (B_M B_{\Delta W} + L_\sigma B_J B_{\Delta Z} B_W) \sum_{i=0}^{L-\ell-1} B_P^{i+1} B_W^i C_\sigma^i
\end{aligned}$$

As a result,

$$\begin{aligned}
s^{(t,\ell)} \leq & B_M B_{\Delta W} + B_W B_J L_\sigma B_{\Delta Z} + B_W C_\sigma \|\widetilde{J}^{(t,\ell)} - \widetilde{J}^{(t-1,\ell)}\|_F \\
= & (B_M B_{\Delta W} + B_W B_J L_\sigma B_{\Delta Z}) + B_P^{(L-\ell)} B_W^{(L-\ell+1)} C_\sigma^{(L-\ell+1)} B_{\Delta H} L_{\text{loss}} \\
& + (B_M B_{\Delta W} + L_\sigma B_J B_{\Delta Z} B_W) \sum_{i=1}^{L-\ell} B_P^i B_W^i C_\sigma^i \\
\leq & B_P^{(L-\ell)} B_W^{(L-\ell+1)} C_\sigma^{(L-\ell+1)} B_{\Delta H} L_{\text{loss}} \\
& + (B_M B_{\Delta W} + L_\sigma B_J B_{\Delta Z} B_W) \sum_{i=0}^{L-\ell} B_P^i B_W^i C_\sigma^i
\end{aligned}$$

$$\begin{aligned}
\|\widetilde{J}^{(t,\ell-1)} - \widetilde{J}^{(t-1,\ell-1)}\|_F = & \left\| P_{in}^\top \left(\widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top - \widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top \right) \right\|_F \\
& + \left\| P_{bd}^\top \left(\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t-2,\ell)} [\widetilde{W}^{(t-2,\ell)}]^\top \right) \right\|_F \\
\leq & B_P s^{(t,\ell)} \\
\leq & (B_P B_W C_\sigma)^{(L-\ell+1)} B_{\Delta H} L_{\text{loss}} \\
& + (B_M B_{\Delta W} + L_\sigma B_J B_{\Delta Z} B_W) \sum_{i=0}^{L-\ell} B_P^{i+1} B_W^i C_\sigma^i
\end{aligned}$$

□

From Equation 5, we can also conclude that

Corollary A.2. *$\|\widetilde{M}^{(t,\ell)} - \widetilde{M}^{(t-1,\ell)}\|_F \leq B_{\Delta M}$ with $B_{\Delta M} = B_J L_\sigma B_{\Delta Z} + C_\sigma B_{\Delta J}$.*

A.3 Bounded Feature Error and Gradient Error

In this subsection, we compare the difference between generic GCN and PipeGCN with the same parameter set, i.e., $\theta = \widetilde{\theta}^{(t)}$.

Lemma A.4. *$\|\widetilde{Z}^{(t,\ell)} - Z^{(\ell)}\|_F \le E_Z$,$\|\widetilde{H}^{(t,\ell)} - H^{(\ell)}\|_F \le E_H$ where $E_Z = B_{\Delta H} \sum\limits_{i=1}^{L} C_\sigma^{i-1} B_W^i B_P^i$ and $E_H = B_{\Delta H} \sum\limits_{i=1}^{L} (C_\sigma B_W B_P)^i$.*

Proof.

$$\begin{aligned}
\|\widetilde{Z}^{(t,\ell)} - Z^{(\ell)}\|_F &= \|(P_{in}\widetilde{H}^{(t,\ell-1)}\widetilde{W}^{(t,\ell)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)}\widetilde{W}^{(t,\ell)}) - (PH^{(\ell-1)}W^{(\ell)})\|_F \\
&\le \|(P_{in}\widetilde{H}^{(t,\ell-1)} + P_{bd}\widetilde{H}^{(t-1,\ell-1)} - PH^{(\ell-1)})W^{(\ell)}\|_F \\
&= B_W\|P(\widetilde{H}^{(t,\ell-1)} - H^{(\ell-1)}) + P_{bd}(\widetilde{H}^{(t-1,\ell-1)} - \widetilde{H}^{(t,\ell-1)})\|_F \\
&\le B_W B_P \left(\|\widetilde{H}^{(t,\ell-1)} - H^{(\ell-1)}\|_F + B_{\Delta H}\right)
\end{aligned}$$

By induction, we assume that $\|\widetilde{H}^{(t,\ell-1)} - H^{(\ell-1)}\|_F \le B_{\Delta H} \sum\limits_{i=1}^{\ell-1} (C_\sigma B_W B_P)^i$. Therefore,

$$\begin{aligned}
\|\widetilde{Z}^{(t,\ell)} - Z^{(\ell)}\|_F &\le B_W B_P B_{\Delta H} \sum_{i=0}^{\ell-1} (C_\sigma B_W B_P)^i \\
&= B_{\Delta H} \sum_{i=1}^{\ell} C_\sigma^{i-1} B_W^i B_P^i
\end{aligned}$$

$$\begin{aligned}
\|\widetilde{H}^{(t,\ell)} - H^{(\ell)}\|_F &= \|\sigma(\widetilde{Z}^{(t,\ell)}) - \sigma(Z^{(\ell)})\|_F \\
&\le C_\sigma \|\widetilde{Z}^{(t,\ell)} - Z^{(\ell)}\|_F \\
&\le B_{\Delta H} \sum_{i=1}^{\ell} (C_\sigma B_W B_P)^i
\end{aligned}$$

□

Lemma A.5. *$\|\widetilde{J}^{(t,\ell)} - J^{(\ell)}\|_F \le E_J$ and $\|\widetilde{M}^{(t,\ell)} - M^{(\ell)}\|_F \le E_M$ with*

$$E_J = \max_{\ell} (B_P B_W C_\sigma)^\ell L_{loss} E_H + B_P(B_W(B_J E_Z L_\sigma + B_{\Delta M}) + B_{\Delta W} B_M) \sum_{i=0}^{L-1} (B_P B_W C_\sigma)^i$$

$$E_M = C_\sigma E_J + L_\sigma B_J E_Z$$

Proof. When $\ell = L$, $\|\widetilde{J}^{(t,L)} - J^{(L)}\|_F \le L_{\text{loss}} E_H$. We assume that

$$\|\widetilde{J}^{(t,\ell)} - J^{(\ell)}\|_F \le (B_P B_W C_\sigma)^{L-\ell} L_{\text{loss}} E_H + U \sum_{i=0}^{L-\ell-1} (B_P B_W C_\sigma)^i \tag{6}$$

$$\|\widetilde{M}^{(t,\ell)} - M^{(\ell)}\|_F \le (B_P B_W C_\sigma)^{L-\ell} C_\sigma L_{\text{loss}} E_H + U C_\sigma \sum_{i=0}^{L-\ell-1} (B_P B_W C_\sigma)^i + L_\sigma B_J E_Z \tag{7}$$

where $U = B_P(B_W B_J E_Z L_\sigma + B_{\Delta W} B_M + B_W B_{\Delta M})$. We prove them by induction as follows.

$$\begin{aligned}
&\|\widetilde{M}^{(t,\ell)} - M^{(\ell)}\|_F \\
&= \|\widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t,\ell)}) - J^{(\ell)} \circ \sigma'(Z^{(\ell)})\|_F \\
&\le \|\widetilde{J}^{(t,\ell)} \circ \sigma'(\widetilde{Z}^{(t,\ell)}) - \widetilde{J}^{(t,\ell)} \circ \sigma'(Z^{(\ell)})\|_F + \|\widetilde{J}^{(t,\ell)} \circ \sigma'(Z^{(\ell)}) - J^{(\ell)} \circ \sigma'(Z^{(\ell)})\|_F \\
&\le B_J\|\sigma'(\widetilde{Z}^{(t,\ell)}) - \sigma'(Z^{(\ell)})\|_F + C_\sigma\|\widetilde{J}^{(t,\ell)} - J^{(\ell)}\|_F
\end{aligned}$$

Here $\|\sigma'(\widetilde{Z}^{(t,\ell)}) - \sigma'(Z^{(\ell)})\|_F \leq L_\sigma E_Z$. With Equation 6,

$$\|\widetilde{M}^{(t,\ell)} - M^{(\ell)}\|_F \leq (B_P B_W C_\sigma)^{L-\ell} C_\sigma L_{\text{loss}} E_H + U C_\sigma \sum_{i=0}^{L-\ell-1} (B_P B_W C_\sigma)^i + L_\sigma B_J E_Z$$

On the other hand,

$$\begin{aligned}
&\|\widetilde{J}^{(t,\ell-1)} - J^{(\ell-1)}\|_F \\
&= \|P_{in}^\top \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top + P_{bd}^\top \widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - P^\top M^{(\ell)} [W^{(\ell)}]^\top \|_F \\
&= \|P^\top (\widetilde{M}^{(t,\ell)} - M^{(\ell)}) [W^{(\ell)}]^\top + P_{bd}^\top (\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top)\|_F \\
&\leq \|P^\top (\widetilde{M}^{(t,\ell)} - M^{(\ell)}) [W^{(\ell)}]^\top\|_F + \|P_{bd}^\top (\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top)\|_F \\
&\leq B_P B_W \|\widetilde{M}^{(t,\ell)} - M^{(\ell)}\|_F + B_P \|\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top\|_F
\end{aligned}$$

The first part is bounded by Equation 7. For the second part,

$$\begin{aligned}
&\|\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top\|_F \\
&\leq \|\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t,\ell)}]^\top\|_F + \|\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t,\ell)}]^\top - \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top\|_F \\
&\leq B_{\Delta W} B_M + B_W B_{\Delta M}
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\|\widetilde{J}^{(t,\ell-1)} - J^{(\ell-1)}\|_F \\
&\leq B_P B_W \|\widetilde{M}^{(t,\ell)} - M^{(\ell)}\|_F + B_P \|\widetilde{M}^{(t-1,\ell)} [\widetilde{W}^{(t-1,\ell)}]^\top - \widetilde{M}^{(t,\ell)} [\widetilde{W}^{(t,\ell)}]^\top\|_F \\
&\leq (B_P B_W C_\sigma)^{L-\ell+1} L_{\text{loss}} E_H + U \sum_{i=1}^{L-\ell} (B_P B_W C_\sigma)^i + U \\
&= (B_P B_W C_\sigma)^{L-\ell+1} L_{\text{loss}} E_H + U \sum_{i=0}^{L-\ell} (B_P B_W C_\sigma)^i
\end{aligned}$$

□

Lemma A.6. $\|\widetilde{G}^{(t,\ell)} - G^{(\ell)}\|_F \leq E_G$ *where* $E_G = B_P(B_H E_M + B_M E_H)$

Proof.

$$\begin{aligned}
&\|\widetilde{G}^{(t,\ell)} - G^{(\ell)}\|_F \\
=& \left\| [P_{in} \widetilde{H}^{(t,\ell-1)} + P_{bd} \widetilde{H}^{(t-1,\ell-1)}]^\top \widetilde{M}^{(t,\ell)} - [P H^{(\ell)}]^\top M^{(\ell)} \right\|_F \\
\leq& \left\| [P_{in} \widetilde{H}^{(t,\ell-1)} + P_{bd} \widetilde{H}^{(t-1,\ell-1)}]^\top \widetilde{M}^{(t,\ell)} - [P H^{(\ell-1)}]^\top \widetilde{M}^{(t,\ell)} \right\|_F \\
&+ \left\| [P H^{(\ell-1)}]^\top \widetilde{M}^{(t,\ell)} - [P H^{(\ell-1)}]^\top M^{(\ell)} \right\|_F \\
\leq& B_M (\|P(\widetilde{H}^{(t,\ell-1)} - H^{(\ell-1)}) + P_{bd}(\widetilde{H}^{(t-1,\ell-1)} - \widetilde{H}^{(t,\ell-1)})\|_F) + B_P B_H E_M \\
\leq& B_M B_P (E_H + B_{\Delta H}) + B_P B_H E_M
\end{aligned}$$

□

By summing up from $\ell = 1$ to $\ell = L$ to both sides, we have

Corollary A.3. $\|\nabla \widetilde{\mathcal{L}}(\theta) - \nabla \mathcal{L}(\theta)\|_2 \leq E_{loss} \coloneqq L E_G$.

According to the derivation of E_{loss}, we observe that E_{loss} contains a factor η. We can rewrite E_{loss}.

Corollary A.4. $\|\nabla \widetilde{\mathcal{L}}(\theta) - \nabla \mathcal{L}(\theta)\|_2 \leq \eta E$ *where*

$$U = B_P B_W C_\sigma, U_m = \max\{1, U^L\}, S = U_m^4 B_X^2 B_P^2 C_\sigma C_{loss} U \left(\sum_{i=0}^{L-1} U^i \right)^2$$

$$R = C_{loss}\left(2L_\sigma S + U_m^2 C_\sigma^2 C_{loss} B_X \sum_{i=0}^{L-1} U^i\right)$$

$$E = LB_P\left(U_m B_X\left(C_\sigma^2 S(C_{loss} + L_{loss}) + R\sum_{i=0}^{L} U^i\right) + C_\sigma^2 C_{loss} S\right)$$

A.4 Proof of the Main Theorem

We first introduce a lemma before the proof of our main theorem.

Lemma A.7 (Lemma 1 in (Cong et al., 2021))**.** *An L-layer GCN is L_f-Lipschitz smoothness, i.e.,* $\|\nabla\mathcal{L}(\theta_1) - \nabla\mathcal{L}(\theta_2)\|_2 \le L_f\|\theta_1 - \theta_2\|_2$.

Now we prove the main theorem.

Proof. With the smoothness of the model,

$$\begin{aligned}\mathcal{L}(\theta^{(t+1)}) &\le \mathcal{L}(\theta^{(t)}) + \left\langle \nabla\mathcal{L}(\theta^{(t)}), \theta^{(t+1)} - \theta^{(t)}\right\rangle + \frac{L_f}{2}\|\theta^{(t+1)} - \theta^{(t)}\|_2^2 \\ &= \mathcal{L}(\theta^{(t)}) - \eta\left\langle \nabla\mathcal{L}(\theta^{(t)}), \nabla\widetilde{\mathcal{L}}(\theta^{(t)})\right\rangle + \frac{\eta^2 L_f}{2}\|\nabla\widetilde{\mathcal{L}}(\theta^{(t)})\|_2^2\end{aligned}$$

Let $\delta^{(t)} = \nabla\widetilde{\mathcal{L}}(\theta^{(t)}) - \nabla\mathcal{L}(\theta^{(t)})$ and $\eta \le 1/L_f$, we have

$$\begin{aligned}\mathcal{L}(\theta^{(t+1)}) &\le \mathcal{L}(\theta^{(t)}) - \eta\left\langle \nabla\mathcal{L}(\theta^{(t)}), \nabla\mathcal{L}(\theta^{(t)}) + \delta^{(t)}\right\rangle + \frac{\eta}{2}\|\nabla\mathcal{L}(\theta^{(t)}) + \delta^{(t)}\|_2^2 \\ &\le \mathcal{L}(\theta^{(t)}) - \frac{\eta}{2}\|\nabla\mathcal{L}(\theta^{(t)})\|_2^2 + \frac{\eta}{2}\|\delta^{(t)}\|_2^2\end{aligned}$$

From Corollary A.4 we know that $\|\delta^{(t)}\|_2 < \eta E$. After rearranging the terms,

$$\|\nabla\mathcal{L}(\theta^{(t)})\|_2^2 \le \frac{2}{\eta}(\mathcal{L}(\theta^{(t)}) - \mathcal{L}(\theta^{(t+1)})) + \eta^2 E^2$$

Summing up from $t = 1$ to T and taking the average,

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^{T}\|\nabla\mathcal{L}(\theta^{(t)})\|_2^2 &\le \frac{2}{\eta T}(\mathcal{L}(\theta^{(1)}) - \mathcal{L}(\theta^{(T+1)})) + \eta^2 E^2 \\ &\le \frac{2}{\eta T}(\mathcal{L}(\theta^{(1)}) - \mathcal{L}(\theta^{*})) + \eta^2 E^2\end{aligned}$$

where θ^* is the minimum point of $\mathcal{L}(\cdot)$. By taking $\eta = \frac{\sqrt{\varepsilon}}{E}$ and $T = (\mathcal{L}(\theta^{(1)}) - \mathcal{L}(\theta^*))E\varepsilon^{-\frac{3}{2}}$ with an arbitrarily small constant $\varepsilon > 0$, we have

$$\frac{1}{T}\sum_{t=1}^{T}\|\nabla\mathcal{L}(\theta^{(t)})\|_2 \le 3\varepsilon$$

□

B Improving Training Throughput over Full-Graph Training Methods (Additional Experiments)

Figure 8 compares the training throughput between PipeGCN and the SOTA full-graph training methods (ROC (Jia et al., 2020) and CAGNET (Tripathy et al., 2020)) on more datasets under that same setting of Figure 3 of the main content. As can be seen, ***the advantage of PipeGCN consistently holds***, which is similar to Figure 3 of the main content.

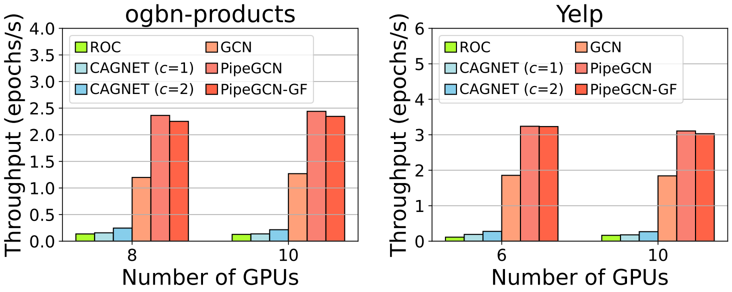


Figure 8: Throughput comparison on ogbn-products and Yelp. Each partition uses one GPU (except CAGNET (c=2) uses two).

C Maintaining Convergence Speed (Additional Experiments)

We provide the additional convergence curves on Yelp in Figure 9. We can see that ***PipeGCN and its variants maintain the convergence speed w.r.t the number of epochs while substantially reducing the end-to-end training time.***

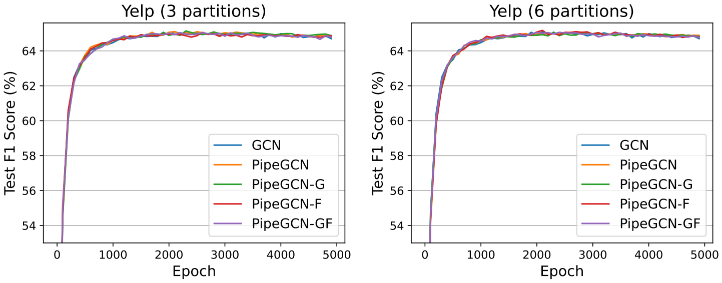


Figure 9: The epoch-to-accuracy comparison on "Yelp" among the vanilla partition-parallel training (GCN) and PipeGCN variants (PipeGCN*), where *PipeGCN and its variants achieve a similar convergence as the vanilla training (without staleness) but are twice as fast in terms of wall-clock time* (see the Throughput improvement in Table 3 of the main content).

D Training Time Improvement Breakdown (Additional Experiments)

To understand the training time improvement offered by PipeGCN, we further breakdown the epoch time into three parts (intra-partition computation, inter-partition communication, and reduce for aggregating model gradient) and provide the result in Figure 10. We can observe that: 1) inter-partition ***communication dominates the training time*** in vanilla distributed training (see GCN); 2) ***PipeGCN (with or without smoothing) greatly hides the communication overhead*** across different number of partitions and all datasets, e.g., the communication time is hidden completely in 2-partition Reddit and almost completely in 3-partition Yelp, thus the substantial reduction in training time; and 3) the proposed ***smoothing incurs only minimal overhead*** (i.e., minor difference between PipeGCN and PipeGCN-GF). Lastly, we also notice that *when communication ratio is extremely large (85%+), PipeGCN hides communication significantly but not completely* (e.g., 10-partition ogbn-products), in which case we can employ those *compression or quantization* techniques from the area of general distributed SGD for further reducing the communication, as the compression is orthogonal to the pipeline method. Besides compression, we can also increase the pipeline depth of PipeGCN, e.g., using two iterations of compute to hide one iteration of communication, which is left to our future work.

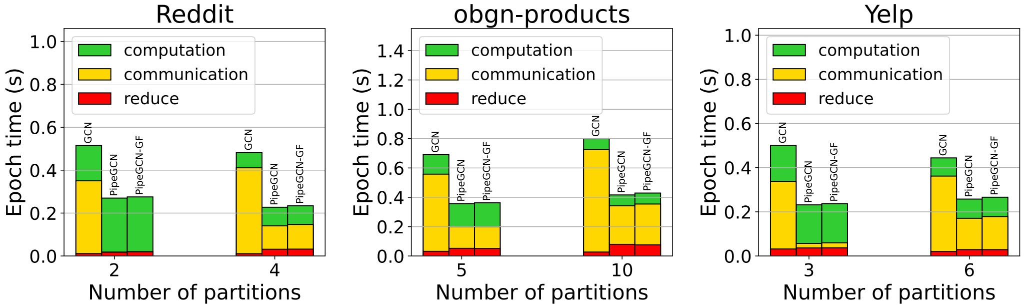


Figure 10: Training time breakdown of vanilla partition-parallel training (GCN), PipeGCN, and PipeGCN with smoothing (PipeGCN-GF).

E PipeGCN with Multiple Computational Nodes

We evaluate PipeGCN on Reddit over different number of partitions and multiple nodes (each contains AMD Radeon Instinct MI60 GPUs, an AMD EPYC 7642 CPU, and 48 lane PCI 3.0 connecting CPU-GPU and GPU-GPU) networked with 10Gbps Ethernet.

The corresponding accuracy results of PipeGCN and its variants are summarized below:

Table 6: The accuracy of PipeGCN and its variants on Reddit.

#partitions (#node*#gpus)	PipeGCN	PipeGCN-F	PipeGCN-G	PipeGCN-GF
2 (1*2)	97.12%	97.09%	97.14%	97.12%
3 (1*3)	97.01%	97.15%	97.17%	97.14%
4 (1*4)	97.04%	97.10%	97.09%	97.10%
6 (2*3)	97.09%	97.12%	97.08%	97.10%
8 (2*4)	97.02%	97.06%	97.15%	97.03%
9 (3*3)	97.03%	97.08%	97.11%	97.08%
12 (3*4)	97.05%	97.05%	97.12%	97.10%
16 (4*4)	96.99%	97.02%	97.14%	97.12%

Furthermore, we provide the speedup against vanilla distributed GCN training below:

Table 7: The speedup of PipeGCN and its vatiants against vanilla distributed GCN training on Reddit.

#nodes*#gpus	GCN	PipeGCN	PipeGCN-G	PipeGCN-F	PipeGCN-GF
1*2	1.00x	1.16x	1.16x	1.16x	1.16x
1*3	1.00x	1.22x	1.22x	1.22x	1.22x
1*4	1.00x	1.29x	1.28x	1.29x	1.28x
2*2	1.00x	1.61x	1.60x	1.61x	1.60x
2*3	1.00x	1.64x	1.64x	1.64x	1.64x
2*4	1.00x	1.41x	1.42x	1.41x	1.37x
3*2	1.00x	1.65x	1.65x	1.65x	1.65x
3*3	1.00x	1.48x	1.49x	1.50x	1.48x
3*4	1.00x	1.35x	1.36x	1.35x	1.34x
4*2	1.00x	1.64x	1.63x	1.63x	1.62x
4*3	1.00x	1.38x	1.38x	1.38x	1.38x
4*4	1.00x	1.30x	1.29x	1.29x	1.29x

From the two tables above, we can observe that our PipeGCN family consistently **maintains the accuracy** of the full-graph training, while **improving the throughput by 15%∼66%** regardless of the machine settings and number of partitions.

F Training Time Breakdown Comparison with Baselines

To understand where PipeGCN gains significantly over baseline algorithms, we provide the detailed time breakdown of ROC and CAGNET on Reddit with the same model in Table 3 (4-layer GraphSAGE, 256 hidden units) in Table 8, in which 'Dist GCN' is the vanilla distributed GCN training illustrated in Figure 1 (a). We observe that PipeGCN greatly saves communication time.

Table 8: Epoch time breakdown of PipeGCN, ROC and CAGNET.

	Total time (s)	Compute (s)	Communication (s)	Reduce (s)
ROC (2 GPUs)	3.63	0.5	3.13	0.00
CAGNET (c=1, 2 GPUs)	2.74	1.91	0.65	0.18
CAGNET (c=2, 2 GPUs)	5.41	4.36	0.09	0.96
Dist GCN (2 GPUs)	0.52	0.17	0.34	0.01
PipeGCN (2 GPUs)	0.27	0.25	0.00	0.02
ROC (4 GPUs)	3.34	0.42	2.92	0.00
CAGNET (c=1, 4 GPUs)	2.31	0.97	1.23	0.11
CAGNET (c=2, 4 GPUs)	2.26	1.03	0.55	0.68
Dist GCN (4 GPUs)	0.48	0.07	0.40	0.01
PipeGCN (4 GPUs)	0.23	0.10	0.10	0.03

G Implementation Details

We discuss the details of the effective and efficient implementation of PipeGCN in this section.

First, for parallel communication and computation, a second cudaStream is required for communication besides the default cudaStream for computation. To also save memory buffers for communication, we batch all communication (e.g., from different layers) into this second cudaStream. When the popular communication backend, Gloo, is used, we parallelize the CPU-GPU transfer with CPU-CPU transfer.

Second, when Dropout layer is used in GCN model, it should be applied after communication. The implementation of the dropout layer for PipeGCN should be considered carefully so that the dropout mask remains consistent for the input tensor and corresponding gradient. If the input feature passes through the dropout layer before being communicated, during the backward phase, the dropout mask is changed and the gradient of masked values is involved in the computation, which introduces noise to the calculation of followup gradients. As a result, the dropout layer can only be applied after receiving boundary features.