Distributed Graph Neural Network Training with Periodic Historical Embedding Synchronization

Zheng Chai∗
Department of Computer Science
George Mason University
Fairfax, VA
zchai2@gmu.edu

Guangji Bai∗
Department of Computer Science
Emory University
Atlanta, GA
guangji.bai@emory.edu

Liang Zhao
Department of Computer Science
Emory University
Atlanta, GA
liang.zhao@emory.edu

Yue Cheng†
Department of Computer Science
George Mason University
Fairfax, VA
yuecheng@gmu.edu

June 2, 2022

ABSTRACT
Despite the recent success of Graph Neural Networks (GNNs), it remains challenging to train a GNN on large graphs (e.g., with over millions of nodes & billions of edges), which are prevalent in various graph-based applications such as social networks, recommender systems, and knowledge graphs. Traditional sampling-based methods accelerate GNN by dropping edges and nodes, which impairs the graph integrity and model performance. Differently, distributed GNN algorithms, which accelerate GNN training by utilizing multiple computing devices, can be classified into two types: "partition-based" methods enjoy low communication cost but suffer from information loss due to dropped edges, while "propagation-based" methods avoid information loss but suffer prohibitive communication overhead caused by neighbor explosion. To jointly address these problems, this paper proposes DIstributed Graph Embedding SynchronizaTion (DIGEST), a novel distributed GNN training framework that synergizes the complementary strength of both categories of existing methods. During subgraph parallel training, we propose to let each device store the historical embedding of its neighbors in other subgraphs. Therefore, our method does not discard any neighbors in other subgraphs (which leads to information loss), nor does it updates them intensively (which leads to communication cost). This effectively avoids (1) the intensive computation on explosively-increasing neighbors and (2) excessive communications across different devices. We proved that the approximation error induced by the staleness of historical embedding can be upper bounded and it does NOT affect the GNN model’s expressiveness. More importantly, our convergence analysis demonstrates that DIGEST enjoys the state-of-the-art convergence rate. Extensive experimental evaluation on large, real-world graph datasets shows that DIGEST achieves up to $21.82 \times$ speedup without compromising the performance compared to state-of-the-art distributed GNN training frameworks.

1 Introduction
Graph Neural Networks (GNNs) have shown impressive success in analyzing non-Euclidean graph data and have achieved promising results in various applications, including social networks, recommender systems and knowledge graphs, etc. [8, 28, 9, 18, 31]. Despite the great promise of GNNs, they meet significant challenges when started being applied to large graphs, which are common in real world—typically the number of nodes of a large graph can be up to millions or even billions. For instance, Facebook social network graph contains over 2.9 billion users and

∗Equal contribution.
†Corresponding author.
over 400 billion friendship relations among users. Amazon needs to recommend 350 million items to 300 million users. Further, natural language processing tasks take advantage of knowledge graphs, such as Freebase [3] with over 1.9 billion triples. Training GNNs on large graphs is jointly challenged by the lack of inherent parallelism of the backpropagation optimization and inter-dependencies among nodes, rendering existing parallel techniques ineffective.

To tackle the unique challenges in GNN training, distributed GNN training is a promising open domain that has attracted fast-increasing attention in recent years. A classic and intuitive way is by sampling. Until now, a good number of graph-sampling-based GNN methods have been proposed, including neighbor-sampling-based methods (e.g., GraphSAGE [12], VR-GCN [4]) and subgraph-sampling-based methods (e.g., Cluster-GCN [6], GraphSAINT [29]). These methods enable a GNN model to be trained over large graphs on a single machine by sampling a subset of data during forward or backward propagation. While sampling operations reduce the size of data needed for computation, these methods suffer from degenerated performance due to unnecessary information loss. To work around this drawback and also to leverage the increasingly powerful computing capability, many recent solutions focus on increasing the number of computing devises and training GNNs using model parallelism in a distributed setup. Distributed GNN training has become a de facto standard for fast and accurate training for Computer Vision and NLP [17, 18, 19].

Existing methods in distributed training for GNNs can be classified into two categories, namely "partition-based" and "propagation-based", by how they tackle the trade-off between communication cost and information loss. "Partition-based" methods [1, 14, 22] partition the graph into different subgraphs by dropping the edges across subgraphs. This way, a big training task on a large graph is decomposed into many smaller training tasks on many subgraphs in parallel, reducing communications among subgraphs, and thus, tasks, due to edge dropping. However, this will result in severe information loss due to the ignorance of the dependencies between nodes and cause performance degeneration. To alleviate information loss, "propagation-based" methods [20, 31, 30, 23, 25] do not ignore the edges across different subgraphs with neighbor communications among subgraphs to satisfy GNN’s neighbor aggregation. However, the number of neighbors involved in neighbor aggregation grows exponentially as the GNN goes deeper (i.e., neighborhood explosion [12]), hence inevitably suffering huge communication overhead and plagued training efficiency.

Therefore, although "partition-based" can decompose the large raw graph into smaller ones and train in parallel, it suffers from severe information loss and low accuracy. "Propagation-based", on the other hand, uses the entire graph for training without information loss but suffers from huge communication overhead and poor efficiency. Hence, it is highly imperative to develop a method that can jointly address both the high communication cost and the severe information loss. Moreover, theoretical guarantees (e.g., on convergence, approximation error) are not well explored for distributed GNN training due to the joint sophistication of graph structure and neural network optimization.

To address the challenges mentioned above, we propose a novel distributed GNN training framework that synergizes the complementary strengths of both partitioning-based and propagating-based methods, named DIstributed Graph Embedding SynchronizaTion (DIGEST). DIGEST does not completely discard node information from other subgraphs to avoid unnecessary information loss, nor does it try to update all the node information to minimize communication costs. Instead, DIGEST uses each device to store the historical embedding of neighbors from other subgraphs. This effectively prevents neighbor updating explosion and avoids huge communication costs across different devices. Furthermore, we proved that the approximation error induced by the staleness of historical embedding can be upper bounded and does NOT affect the GNN model’s expressiveness. More importantly, global convergence guarantee is provided, which demonstrates that DIGEST has the state-of-the-art convergence rate. The main contributions of this paper can be summarized as:

- **Proposing a novel distributed GNN training framework that integrates the advantages of partition-based and communication-based methods.** Existing methods in distributed GNN training focus on two different but complementary aspects. Partition-based methods focus on minimizing the communication cost among different subgraphs while propagation-based methods aim at preserving information of the raw graph data. DIGEST drops no edges while avoiding communication overhead by integrating the strength of both categories of methods.

- **Developing a periodic historical embedding synchronization technique for distributed GNN training.** DIGEST utilizes the entire graph data for training by separating in-subgraph and out-of-subgraph neighbor nodes and approximating the latter with historical embeddings. Instead of issuing frequent exchanges of embeddings between the subgraphs during training, DIGEST pulls/pushes historical embeddings before or after the training periodically to keep the communication costs low.

- **Providing extensive theoretical guarantee on both performance and convergence of the proposed algorithm.** We proved that DIGEST’s convergence rate is $O(T^{-2/3}M^{-1/3})$, with $T$ iterations and $M$ subgraphs, which is close to vanilla distributed GNN training without staleness. We also showed the upper bound on the approximation error of the staleness and further demonstrated that DIGEST can achieve equivalent expressiveness to WL test [27].

- **Conducting comprehensive empirical results on both performance and speedup.** We perform extensive evaluation on four benchmark with classic GNNs (e.g., GCN [12] and GAT [24]). The experimental results show that for the
best case DIGEST improves the performance by 33.14\%, and achieves 21.82× speedup in training time compared to two state-of-the-art distributed GNNs training frameworks.

2 Background and problem formulation

In this section, we first introduce the Graph Neural Network (GNN) and its training on a single machine, and then formulate the problem of distributed GNN training.

**Graph Neural Networks.** GNNs aim to learn a function of signals/features on a graph \(G(V, E)\) with node features \(X \in \mathbb{R}^{|V| \times d}\), where \(d\) denotes the node feature dimension. For typical semi-supervised node classification tasks \([16]\), where each node \(v \in V\) is associated with a label \(y_v\), a \(L\)-layer GNN \(f\) is trained to learn the node representation \(h_v\) such that \(y_v\) can be predicted accurately. The training process of a GNN can be practically described as the node representation learning based on the *message passing mechanism* \([11]\). Analytically, given a graph \(G(V, E)\) and a node \(v \in V\), the \((\ell + 1)\)-th layer of the GNN is defined as

\[
\mathbf{h}^{(\ell+1)}_v = f^{(\ell+1)}(\mathbf{h}^{(\ell)}_v, \{\mathbf{h}^{(\ell)}_u : u \in N(v)\}) = \Psi^{(\ell+1)}(\mathbf{h}^{(\ell)}_v, \Phi^{(\ell+1)}(\{\mathbf{h}^{(\ell)}_u : u \in N(v)\}))
\]

where \(\mathbf{h}^{(\ell)}_v\) represents the embedding of node \(v\) obtained in the \(\ell\)-th layer, with \(0 \leq \ell < L\) and \(\mathbf{h}^{(0)}_v\) being initialized to \(x_v\) (\(v\)-th row in \(X\)), and \(N(v)\) represents the set of direct 1-hop neighbors for node \(v\). Each layer of the GNN, i.e. \(f^{(\ell)}\), can be further decomposed into two components: 1) Aggregation function \(\Phi^{(\ell)}\), which takes the nodes embeddings of node \(v\)’s neighbors as input, and output the aggregated neighborhood embedding. 2) Updating function \(\Psi^{(\ell)}\), which combines the embedding of \(v\) and the aggregated neighborhood embedding to update the embedding of node \(v\) for the next layer. Both \(\Phi^{(\ell)}\) and \(\Psi^{(\ell)}\) can choose to use various functions in different types of GNNs. For example, GraphSAGE \([12]\) samples a subset of neighbors with certain size and provides three distinct aggregation algorithms: mean aggregator, LSTM aggregator, and pooling aggregator, while GCN \([16]\) performs weighted averaging on all neighbor nodes for aggregation.

To train a GNN on a single machine, one can minimize the empirical loss \(L(W)\) over the entire graph in the training data, i.e., \(L(W) = (1/|V|) \sum_{v \in V} \text{Loss}(\mathbf{h}^{(L)}_v, y_v)\), where \(\text{Loss}(\cdot, \cdot)\) denotes a loss function (e.g., cross entropy loss), and \(\mathbf{h}^{(L)}_v\) denotes the embedding of node \(v\) from the last layer of the GNN and can be calculated by following Eq. \(1\) recursively.

**Distributed Training for GNNs.** Distributed GNN training means to first partition the original graph into multiple subgraphs without overlap, which can also be considered as mini batches. Then different mini-batches are trained in different devices in parallel. Here, Eq. \(1\) can be further reformulated as

\[
\mathbf{h}^{(\ell+1)}_v = \Psi^{(\ell+1)}\left(\mathbf{h}^{(\ell)}_v, \Phi^{(\ell+1)}\left(\left\{\mathbf{h}^{(\ell)}_u : u \in N(v) \cap S(v)\right\} \cup \left\{\mathbf{h}^{(\ell)}_u : u \in N(v) \setminus S(v)\right\}\right)\right)
\]

where \(S(v)\) denotes the subgraph that node \(v\) belongs to. In this paper, we consider the distributed training of GNNs with multiple local machines and a global server. The original input graph \(G\) is first partitioned into \(M\) subgraphs, where each \(G_m(V_m, E_m)\) represents the subgraph \(m\). Our goal is to find the optimal set of parameters \(W\) in a distributed manner by minimizing each local loss, i.e.,

\[
\min_{W} L^{\text{Local}}(W_m) = \frac{1}{|V_m|} \sum_{v \in V_m} \text{Loss}(\mathbf{h}^{(L)}_v, y_v), \quad m = 1, 2, \cdots, M \text{ in parallel}
\]

where \(W_m = \{W^{(L)}_m\}_{\ell=1}^L\) are local parameters and \(\mathbf{h}^{(L)}_v\) follows Eq. \(2\) recursively.

**Challenges.** The main challenges for distributed training of GNNs lie in the trade-off between *communication cost* and *information loss*. "Partition-based" generalizes the existing data parallelism techniques of classical distributed training on *i.i.d.* data to graph data and enjoys minimal communication cost. However, directly partitioning a large graph into multiple subgraphs can result in severe information loss due to the ignorance of huge number of cross-subgraph edges and cause performance degeneration \([11, 14, 22]\). For these methods, the embedding of neighbors out of the current subgraph (second embedding set in Eq. \(2\) are dropped and the connections between subgraphs are thus ignored. Hence, another line of work \([26]\) "propagation-based" considers using communication of neighbor nodes for each subgraph to satisfy GNN’s neighbor aggregation, which minimizes the information loss. As shown in Eq. \(2\), the embeddings for neighbor nodes outside the current subgraph is swapped between different subgraphs. However, the
Figure 1: **Distributed GNN training methods.** (a): Propagation-based methods rely on communication of out-of-subgraph neighbor nodes for exact message passing even in a distributed setup. (b): Partition-based methods decompose the original problem into multiple smaller ones and directly apply data parallelism onto partitioned subgraph data. (c): In DIGEST each device stores the historical embeddings of all its neighbors from other subgraphs. Propagation-based methods suffer high communication cost (red vertical double arrows in (a)) due to neighbor explosion, while partition-based methods suffer severe information loss due to dropped edges (red crosses in (b)). DIGEST combines the best of both worlds. ALL nodes are utilized in DIGEST to achieve full-graph awareness, while periodic historical embedding synchronization keeps the communication cost low.

number of neighbors involved in the neighbor aggregation process expands exponentially as the GNN model goes deep, which is known as the *neighborhood explosion* problem. Hence, though no edges are dropped in this case, inevitable communication overhead is incurred and plagues the achievable training efficiency \[20, 31, 30, 23, 25\]. Moreover, theoretical guarantees (e.g., on convergence, approximation error) are not well explored for distributed GNN due to the joint sophistication of graph structure and neural network optimization.

3 Proposed method

In this section, we introduce the proposed GNN training framework DIGEST. DIGEST leverages *both* types of embeddings in Eq. 2 to address the information loss issue. In addition, instead of exchanging embeddings during the training process between the subgraphs, DIGEST only pull and push the historical embeddings before or after each step of training periodically. With this strategy, the communications turn to be more efficient, which are illustrated in Figure 1 and analyzed in more details in Section 3.3. Moreover, we prove that the error introduced by the staleness of the historical embedding is upper-bounded while the convergence is also guaranteed.

3.1 Distributed GNN training with full-graph awareness

In DIGEST, each copy of GNN trained on a local machine will make use of all available graph information, i.e. *no* edges are dropped in both forward and backward propagation. Analytically, calculating each local gradient \( \nabla L_{m} \) as defined in Eq. 3 will involve out-of-subgraph neighbor information. For out-of-subgraph neighbor nodes, we approximate their embeddings via historical embeddings acquired in previous training, denoted by \( \tilde{h}_{\ell}^{(v)} \). Formally, given a node \( v \in G_{m}(V_{m}, E_{m}) \), the forward propagation for the \((\ell + 1)\)-th layer of DIGEST is achieved by modifying Eq. 2 as

\[
\begin{align*}
    h_{v}^{(\ell+1)} & = F(H_{in}^{(\ell)}, \tilde{H}_{out}^{(\ell)}) \\
    & := \Psi^{(\ell+1)}(h_{v}^{(\ell)}, \Phi^{(\ell+1)}(h_{u}^{(\ell)} : u \in \mathcal{N}(v) \cap V_{m}) \cup \{\tilde{h}_{u}^{(\ell)} : u \in \mathcal{N}(v) \setminus V_{m}\})
\end{align*}
\]

(4)

where \( H_{in}^{(\ell)} \) and \( \tilde{H}_{out}^{(\ell)} \) denotes the set of in-subgraph node embeddings and out-of-subgraph historical embeddings, respectively. \( F \) is defined as the forward propagation function of the GNN for compact formula. As can be seen in
Eq. 5 DIGEST considers all neighbor nodes information during forward propagation. On the other hand, leveraging the entire graph data in forward propagation will in turn improve the estimation of gradient in backpropagation. To see this, we first reformulate $F(H^l_{in}, \tilde{H}^l_{out})$ into the matrix form that specifies the right hand side of Eq. 4

$$F(H^l_{in}, \tilde{H}^l_{out}) = \sigma \left( P_{in}H^l_{in}W^l_{m} + P_{out}\tilde{H}^l_{out}W^l_{m} \right)$$

(5)

where we consider the GCN model as an example for illustration but our analyses apply to general cases of any GNN models. $P_{in}$ and $P_{out}$ denotes the propagation matrix for in-subgraph nodes and out-of-subgraph nodes, respectively, and we have $P = P_{in} + P_{out}$ where $P$ is the original full-graph propagation matrix. $\sigma$ is the activation function following GCN’s definition. Hence,

$$\frac{\partial}{\partial W}F(H^l_{in}, \tilde{H}^l_{out}) = \frac{\partial}{\partial W} \sigma \left( P_{in}H^l_{in}W^l_{m} + P_{out}\tilde{H}^l_{out}W^l_{m} \right)$$

$$= \left[ P_{in}H^l_{in} + P_{out}\tilde{H}^l_{out} \right] \cdot G^l$$

(6)

where

$$G^l = \frac{\partial}{\partial H}F(H^{l+1}_{in}, \tilde{H}^{l+1}_{out}) \circ \sigma' \left( P_{in}H^l_{in}W^l_{m} + P_{out}\tilde{H}^l_{out}W^l_{m} \right)$$

(7)

The key observation here is that ALL neighbor nodes are involved in the backpropagation since $\partial F(H^l_{in}, \tilde{H}^l_{out})/\partial W$ also depends on $\tilde{H}^l_{out}$ (second line in Eq. 6 and also Eq. 7). The separation of in-subgraph nodes and out-of-subgraph nodes, and their approximation via historical embedding form the very foundation of DIGEST.

3.2 Periodic historical embedding synchronization with communication efficiency

DIGEST maintains a storage space for all the historical embeddings, the historical embeddings of layer $\ell$ for all nodes in $\mathcal{V}$ can be formulated as $\tilde{H}^l = \{\tilde{h}_v : v \in \mathcal{V}\}$.

For any subgraph $G_m$, before the start of the current epoch, the necessary historical embeddings $\tilde{H}^l_{out} = \{\tilde{h}_u : u \in \mathcal{N}(v) \setminus \mathcal{V}_m, \forall v \in \mathcal{V}_m\}$ are pulled from the storage space, called "pull" operation denoted as $H^l_{out} \leftarrow \tilde{H}^l_{out}$. After the end of current epoch, the newly computed embeddings $H^l_{out} = \{h^l_v : \forall v \in \mathcal{V}_m\}$ are pushed to the same storage and server as historical embeddings in future epochs, called "push" operation denoted as $H^l_{out} \leftarrow \tilde{H}^l_{out}$. We notice that there are large number of node embeddings involved in the both pull and push operations, and more importantly, nodes are independent of each other on these two operations. Hence it is naturally fit for parallel process in terms of model layer level and node level. For subgraph $G_m$, the total number of historical embeddings needed to be pulled from the central storage is $|\tilde{H}^l_{out}$. Assume that it takes time $t$ to pull for one node, the total time cost should be $|\tilde{H}^l_{out}| \times t$ if processing in serial. But with parallel processing, theoretically we can still keep the pull time for $v \in \mathcal{V}_m$ as $t$.

Another strategy used for improving the communication is applying periodic embedding synchronization instead of synchronizing every epoch. Increasing the frequency of synchronization will benefit performance, but there will be additional time consumption. We explore the relation between performance over time and synchronization frequency of embeddings in Section 5.2.

Algorithm 1 shows the training process of DIGEST. At the beginning of training, the original graph is partitioned into several subgraphs with off-the-shelf graph clustering methods such as the widely-used METIS algorithm [15]. Then the mini-batches are distributed to distinct workers (i.e. GPUs or VMs). Depends on the size of mini-batch, each worker can handle one or more subgraphs. DIGEST has two types of synchronization: Model weights synchronization and historical embedding synchronization as illustrated in Figure 1(c). The former one is performed for each epoch by ALLReduce operation which can aggregates all the model weights of subgraphs into global weights in parallel (Line 13). For the historical embedding synchronization, we synchronize every $N$ epochs to gain the optimal performance over training time with the defined pull and push operations (Lines 5,6,9,10).

3.3 Complexity analysis

Here we analyze the memory and communication complexity of DIGEST, compared with propagation-based methods. Without loss of generality, consider a GNN with $L$ layers and each layer has a fixed width $d$. For propagation-based methods, the memory complexity for each local machine grows exponentially with respect to $L$. In comparison, DIGEST pulls the required out-of-subgraph node embeddings and keeps them locally for each local machine. For the $m$-th local machine and the corresponding subgraph on it, i.e. $G_m(\mathcal{V}_m, \mathcal{E}_m)$, the memory complexity per training iteration is $O(\bigcup_{v \in \mathcal{V}_m} \mathcal{N}(v) \cup \{v\} \cdot L \cdot d)$, which scales linearly with respect to the number of GNN layers. Propagation-based
Algorithm 1 Distributed GNN training with periodic historical embedding synchronization

Input: Graph $G(V, E)$; GNN depth $L$; training epoch $R$; global parameters $W^{(r)} = \{W^{(r, \ell)}\}_{\ell=1}^L$, local parameters $W^{(r, \ell)} = \{W^{(r, \ell)}\}_{\ell=1}^L$, $\forall m \in [M], r \in [R]$; non-linearity activation function $\sigma$; neighborhood function $\mathcal{N} : v \rightarrow 2^V$; synchronization interval $N$; learning rate $\eta$.

Output: The trained model weights $W^{(R+1)}$.

Digest():
1. $W^{(1)} = \{W_{m}^{(1)}\}_{m=1}^M$
2. $\{G_{m}(V_{m}, E_{m}), m = 1, 2, \ldots, M\} \leftarrow \text{METIS}(G)$ \Comment{graph partition}
3. for $r = 1 \ldots R$ do
4.  for $m = 1, \ldots, M$ in parallel do
5.      $W^{(r)}_{m} = W^{(r)}$
6.      for $\ell = 1 \ldots L$ do
7.          $H^{(\ell)}_{m} \leftarrow H^{(\ell)}$
8.      for $v \in V_{m}$ do
9.          $h^{(\ell)}_{v} = \{h^{(\ell)}_{u} : u \in \mathcal{N}(v) \setminus V_{m}\}$
10.         $h^{(\ell)}_{m} = \{h^{(\ell)}_{u} : u \in \mathcal{N}(v) \cap V_{m}\}$
11.         $h^{(\ell)}_{m} = \sigma(W^{(r, \ell)}_{m}, \text{CONCAT}(h^{(\ell)}_{v}, h^{(\ell)}_{in}, h^{(\ell)}_{out}))$
12.      if $(r - 1) \% N == 0$ and $\ell \neq L$ then
13.          $H^{(\ell)}_{m} \leftarrow H^{(\ell)}$
14.          $h^{(\ell)}_{v} \leftarrow h^{(\ell)}_{v} + \|h^{(\ell)}_{v}\|_2, \forall v \in V_{m}$ \Comment{PUSH}
15.      $W^{(r, \ell+1)}_{m} = W^{(r, \ell)}_{m} - \eta \cdot \nabla W^{(r, \ell)}_{m}$ \Comment{update local parameters}
16.      $W^{(r+1)}_{m} \leftarrow \text{AllReduce}(W^{(r+1)}_{1} \ldots W^{(r+1)}_{M})$ \Comment{update global parameters}
17.  return $W^{(R+1)}$

In this section, we provide theoretical analyses of the propose distributed strategy DIGEST, including the key issues on bound of error induced by the staleness of embeddings and gradients, equivalent expressiveness to the Weisfeiler-Lehman (WL) test, and convergence guarantee. Our analysis is generic for arbitrary GNN $f$ with Lipschitz-smooth aggregating function $\Phi$ and updating function $\Psi$. All proofs can be found in the appendix.

4 Theoretical analysis

4.1 Error bound on approximated embedding and gradients

Theorem 1. Given a $L$-layer GNN $f_W$ with $r_1$-Lipschitz continuous $\Phi$ and $r_2$-Lipschitz continuous $\Psi$. If $\forall v \in V$ and $\forall \ell \in \{1, 2, \ldots, L-1\}$ we have $\|h^{(\ell)}_{v} - h^{(\ell)}_{v}\|_2 \leq \epsilon^{(\ell)}$, then the embedding from the last layer (i.e., final output) of the GNN is bounded by $\|h^{(L)}_{v} - h^{(L)}_{v}\|_2 \leq \sum_{\ell=1}^{L-1} \epsilon^{(\ell)} r_1^{L-\ell} r_2^{L-\ell} |\mathcal{N}(v)|^{L-\ell}$, where $h^{(L)}_{v}$ denotes the exact output from the last layer of GNN without any staleness.

Following Theorem 1, we can derive the upper bound for gradients as well:

Corollary 1. Given a $L$-layer GNN $f_W$ with $r_1$-Lipschitz continuous $\Phi$ and $r_2$-Lipschitz continuous $\Psi$. If $\forall v \in V$ and $\forall \ell \in \{1, 2, \ldots, L-1\}$ we have $\|h^{(\ell)}_{v} - h^{(\ell)}_{v}\|_2 \leq \epsilon^{(\ell)}$, then the gradients of GNN can be upper bounded by
\[ \| \nabla W L^{\text{Local}}_m (h_v^{(L)}) - \nabla W L^{\text{Local}}_m (h_\ast_v^{(L)}) \| \leq \tau \| h_v^{(L)} - h_\ast_v^{(L)} \|, \] if node \( v \) is within the \( m \)-th subgraph and the loss function \( L^{\text{Local}}_m \) is \( \tau \)-Lipschitz smooth.

### 4.2 Equivalency to WL test in expressiveness

**Lemma 1.** Given countable embedding set \{ \( h_v^{(L-1)} : v \in \mathcal{V} \), s.t. \( \| h_v^{(L-1)} - h_w^{(L-1)} \| > 2(\mu + \epsilon), \forall v \neq w \in \mathcal{V} \). If \( \| h_v^{(L-1)} - h_\ast_v^{(L-1)} \| \leq \mu \) and \( \| h_\ast_v^{(L-1)} - \hat{h}_\ast_v^{(L-1)} \| \leq \epsilon \), then \( \exists \Phi \) and \( \Psi \), s.t.

\[
\| f^{(L)}(h_v^{(L-1)}) - f^{(L)}(h_\ast_v^{(L-1)}) \| \leq \mu + \epsilon \text{ and } \| f^{(L)}(h_\ast_v^{(L-1)}) - f^{(L)}(h_\ast_v^{(L-1)}) \| \geq 2(\mu + \epsilon + \lambda) \quad (8)
\]

\( \forall v \neq w \in \mathcal{V} \) and \( \lambda > 0 \). \( f \) is the GNN composed from \( \Phi \) and \( \Psi \). Parameters are omitted here.

**Theorem 2.** Given a \( L \)-layer GNN \( f \) which can be decomposed into \( \Phi \) and \( \Psi \) that satisfy the conditions in Lemma 1,\( \exists \) a function \( \xi : \mathbb{R}^d \rightarrow \Sigma \) s.t. \( \xi(h_v^{(L)}) = c_v^{(L)}, \forall v \in \mathcal{V} \) where \( c_v^{(L)} \) denotes a node’s coloring after \( L \) rounds of refinement and \( d_L \) is the output layer width.

### 4.3 Convergence guarantee

As both fresh inner-subgraph node embeddings and staled out-of-subgraph embeddings are adopted in our algorithm, its convergence rate is still unknown. We have proved the convergence of DIGEST and present the convergence property in the following theorem.

**Theorem 3** (Convergence of DIGEST). There exists a constant \( E \) such that for any arbitrarily small constant \( \epsilon > 0 \), we can choose a learning rate \( \eta = \frac{\sqrt{M}}{E} \) and number of training iterations \( T = (L(W^{(1)}) - L(W^*)) \frac{E}{\sqrt{M}} \epsilon^{-\frac{3}{2}} \) such that \((1/T) \cdot \sum_{t=1}^{T} \| \nabla L(W_t) \|^2 \leq O(T^{-2/3}M^{-1/3})\), where \( W_t \) and \( W^* \) denotes the parameters at iteration \( t \) and the optimal one, respectively.

Hence, the convergence rate of DIGEST is \( O(T^{-2/3}M^{-1/3}) \), which matches the state-of-the-art PipeGCN’s \( O(T^{-2/3}) \) \cite{25} in terms of number of iterations \( T \), while DIGEST further enjoys faster convergence if more computational resources are available (larger \( M \)).

### 5 Experiments

In this section, we evaluate DIGEST and compare DIGEST against two state-of-the-art distributed GNNs training frameworks as baselines in terms of training efficiency and scalability. Considering the distinct training time per epoch between DIGEST and other baselines, we report the F1 scores on validation dataset and training loss over training time, instead of over communication rounds, in the results. This way it makes a fairer comparison in terms of training performance and efficiency.

![Figure 2: Performance comparison of the GCN training frameworks on four benchmark datasets. The top four subfigures show the training loss over training time, and the bottom four subfigures show the global validation F1 scores during the whole training process. (Better seen in color.)](image)
Table 1: Performance comparison of distributed GNNs frameworks. F1 score on validation dataset reported. Speedup is calculated by normalizing per-epoch training time against that of DGL.

| Method | Metric | GCN | OGB-Arxiv | Flickr | Reddit | OGB-Products |
|--------|--------|-----|-----------|--------|--------|--------------|
| LLCG   | F1     | 69.8 ± 0.21 | 50.73 ± 0.15 | 62.09 ± 0.41 | 90.79 ± 0.16 | 68.84 ± 0.22 |
|        | Speedup| 2.35× | 0.88× | 1.47× | 1.396× | 1.787× |
| DGL    | F1     | 69.9 ± 0.11 | 50.9 ± 0.13 | 87.02 ± 0.23 | 91.01 ± 0.12 | 70.34 ± 0.11 |
|        | Speedup| 1×      | 1×      | 1×      | 1×      | 1×      |
| DIGEST | F1     | 72.2 ± 0.23 | 53.78 ± 0.21 | 95.23 ± 0.43 | 91.55 ± 0.11 | 68.35 ± 0.41 |
|        | Speedup| 17.41× | 11.06× | 7.86× | 3.096× | 11.49× |

5.1 Experiment setting

Implementation and Setup. We have implemented DIGEST and other comparison GNNs training methods all in PyTorch [21]. For all the experiments, we simulate a distributed training environment using an EC2 g4dn.4xlarge virtual machine (VM) instance on AWS, which has 8 NVIDIA T4 GPUs, 96 vCPUs, and 384 GB main memory.

Baselines. Recall in Section 2 we categorize existing distributed GNN training into two types of general methods. In evaluation, we choose two state-of-the-art distributed training frameworks, one from each category as the baseline. For the first category, we choose LLCG [22], which partitions a graph into subgraphs and trains each subgraph strictly independently without incurring any communication among subgraphs. LLCG uses a central server to aggregate local models from each device and performs global training using mini-batches with full neighbor information to ensure that the model learns the global structure of the graph. LLCG uses this additional step to reduce the information loss caused by graph partitioning. For the second category, we choose to use DGL [26], which is a commonly-used, distributed GNN training framework. In contrast to LLCG, DGL requires exchanging node embeddings among partitioned subgraphs. DGL requires frequent swap operations with other subgraphs for embeddings during subgraph’s local training in each epoch, and therefore, DGL incurs high communication cost.

Models and datasets. We report the experimental results of training GCN [16] and GAT [24] in a distributed manner spanning multiple GPU devices. But our framework can also be applied, in a straightforward way, to other prominent GNNs, such as PNA [7], GCNII [5]. Four benchmark node classification datasets, OGB-Arxiv [13], Flickr [29], Reddit [29], and OGB-Products [13], are used in our evaluation. Please refer to the Appendix for the summary of the datasets.

5.2 Experimental results

In this section, we evaluate DIGEST, LLCG, and DGL on the four datasets. Due to the page limit, we move parts of our evaluation results to the Appendix.

Efficiency of DIGEST. We first evaluate the training performance of DIGEST. As shown in Figure 2, DIGEST outperforms both LLCG and DGL for all the datasets when performing distributed training on a pure GCN. LLCG performs worst particularly for the Reddit dataset, because in the global server correction of LLCG, only a mini-batch is trained and it is not sufficient to correct the plain GCN. This is also the reason why the authors of LLCG report the performance of a complex model with mixing GCN layers and GraphSAGE layers [22]. DGL achieves good performance on some dataset (e.g., OGB-products) with uniform node sampling strategy and real-time embedding exchanging. However, frequent communication also leads to slow performance increasing for dataset Flickr (Figure 2(b)) and poor performance for all four datasets. DIGEST avoids these issues and therefore achieves satisfying performance over the training time.

We measure the training time per epoch as shown in Figure 3. Since the embedding synchronization is only performed before the start or after the end of local training, DIGEST takes significantly shorter training time per epoch than that
of LLCG and DGL. Furthermore, DIGEST performs periodic synchronization instead of per-epoch synchronization, which further shortens the training time.

Table 1 presents the detailed numbers for the comparison of three frameworks on the four datasets. For all the cases except GAT on OGB-Arxiv, DIGEST achieves leading F1 scores on the validation dataset, demonstrating the efficacy of DIGEST’s design.

**Scalability of DIGEST.** We evaluate the scalability of three frameworks by training a GCN on OGB-Products with varied number of GPUs. We use average training time per epoch against that of DGL with a single GPU to calculate the speedup results. As shown in Figure 4, DIGEST shows the best scalability compared to the other two. The speedup rises with the number of GPUs used during training. We observe a similar trend for DGL, but the relative speedup for DGL is significantly smaller than that for DIGEST, due to the using of historical embeddings instead of real-time embeddings.

**Synchronization frequency.** We next perform a sensitivity analysis by varying the synchronization intervals for OGB-Products to study how the synchronization frequency would affect the training performance. As shown in Figure 5, DIGEST achieves the highest F1 score over training time when configured to perform synchronization of historical embeddings every 10 epochs. A large interval (20) or a small interval (1) results in performance degradation, due to the long term loss of graph information or additional communication cost.

6 Conclusion

There are two general categories in distributed GNN training. Partition-based methods suffer from graph information loss, while propagation-based methods suffer from high communication cost. In this work we present DIGEST, a novel distributed GNN training framework that synergizes the complementary strengths of both methods by leveraging historical embeddings intelligently. We provide rigorous theoretical analysis to prove that DIGEST has competitive convergence rate and expressiveness. Extensive experiments on four benchmark datasets validate our analysis and demonstrate the efficiency and scalability of DIGEST.

References

[1] A. Angerd, K. Balasubramanian, and M. Annavaram. Distributed training of graph convolutional networks using subgraph approximation. arXiv preprint arXiv:2012.04930, 2020.

[2] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, et al. Towards federated learning at scale: System design. Proceedings of Machine Learning and Systems, 1:374–388, 2019.

[3] N. Chah. Freebase-triples: A methodology for processing the freebase data dumps. arXiv preprint arXiv:1712.08707, 2017.

[4] J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. arXiv preprint arXiv:1710.10568, 2017.

[5] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li. Simple and deep graph convolutional networks. In International Conference on Machine Learning, pages 1725–1735. PMLR, 2020.

[6] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh. Cluster-gen: 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, pages 257–266, 2019.

[7] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković. Principal neighbourhood aggregation for graph nets. Advances in Neural Information Processing Systems, 33:13260–13271, 2020.

[8] H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In International conference on machine learning, pages 2702–2711. PMLR, 2016.

[9] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In Proceedings of the 2018 world wide web conference, pages 1775–1784, 2018.

[10] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In International Conference on Machine Learning, pages 3294–3304. PMLR, 2021.

[11] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In International conference on machine learning, pages 1263–1272. PMLR, 2017.
[12] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[13] 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*, 33:22118–22133, 2020.

[14] 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*, 2:187–198, 2020.

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

[16] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[17] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[18] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang. Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 388–396. IEEE, 2019.

[19] B. Y. Lin, C. He, Z. Zeng, H. Wang, Y. Huang, M. Soltanolkotabi, X. Ren, and S. Avestimehr. Fednlp: A research platform for federated learning in natural language processing. *arXiv preprint arXiv:2104.08815*, 2021.

[20] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai. {NeuGraph}: Parallel deep neural network computation on large graphs. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 443–458, 2019.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[22] M. Ramezani, W. Cong, M. Mahdavi, M. T. Kandemir, and A. Sivasubramaniam. Learn locally, correct globally: A distributed algorithm for training graph neural networks. *arXiv preprint arXiv:2111.08202*, 2021.

[23] A. Tripathy, K. Yelick, and A. Buluç. Reducing communication in graph neural network training. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2020.

[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[25] C. Wan, Y. Li, C. R. Wolfe, A. Kyrillidis, N. S. Kim, and Y. Lin. Pipegcn: Efficient full-graph training of graph convolutional networks with pipelined feature communication. *arXiv preprint arXiv:2203.10428*, 2022.

[26] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[27] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[28] 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*, pages 974–983, 2018.

[29] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Graphsaint: Graph sampling based inductive learning method. *arXiv preprint arXiv:1907.04931*, 2019.

[30] 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 *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44. IEEE, 2020.

[31] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou. Aligraph: a comprehensive graph neural network platform. *arXiv preprint arXiv:1902.08730*, 2019.
Appendix

In this section, we describe detailed experimental setup, additional experimental results, and complete proofs. We reuse part of code adopted from GNNAutoscale [10]; the code for this paper is available at: https://anonymous.4open.science/r/DIGEST Please note that the code is subjected to reorganization to improve the readability.

A Experimental Setup Details

As mentioned in Section 5.1, all the experiments are done on an EC2 g4dn.metal virtual machine (VM) instance on AWS, which has 8 NVIDIA T4 GPUs, 96 vCPUs, and 384 GB main memory. Other important information including operation system version, Linux kernel version, and CUDA version is summarized in Table 2. For fair comparison, we use the same optimizer (Adam), learning rate, and graph partition algorithm for all the three frameworks, DIGEST, LLCG, and DGL. For parameters that are unique to both LLCG and DGL, such as the number of neighbors sampled from each layer for each node, we choose the default value for both LLCG and DGL. Each of the three frameworks has a set of parameters that are exclusively unique to that framework; for these exclusive parameters, we tune them in order to achieve the best performance. Please refer to the configuration files under run/conf/model for detailed configuration setups for all the models and datasets.

Table 2: Summary of environmental setup of our testbed.

| OS        | Linux kernel | CUDA | Driver | PyTorch | PyTorch Geometric | PyTorch Sparse |
|-----------|--------------|------|--------|---------|-------------------|----------------|
| Ubuntu 18.04 | 5.4.0        | 11.6 | 510.47.03 | 1.10.0  | 2.0.4             | 0.6.13         |

We use four datasets: OGB-Arxiv [13], Flickr [29], Reddit [29], and OGB-Products [13] for evaluation. The detailed information of these datasets is summarized in Table 3.

Table 3: Summary of dataset statistics.

| Dataset         | # Nodes   | # Edges    | # Features | # Classes | Train % / Validation % / Test % |
|-----------------|-----------|------------|------------|-----------|---------------------------------|
| Flickr          | 89,250    | 899,756    | 500        | 7         | 50% / 25% / 25%                 |
| Reddit          | 232,965   | 23,213,838 | 602        | 41        | 66% / 10% / 24%                |
| OGB-Arxiv       | 169,343   | 2,315,598  | 128        | 40        | 53.7% / 17.6% / 28.7%          |
| OGB-Products    | 2,449,029 | 123,718,280| 100        | 47        | 8% / 2% / 90%                  |

B Additional Experimental Results

B.1 Performance of GAT Training

We first show the learning curves of training GAT with three methods on three different datasets. As shown in Figure 6, for dataset Flickr and Reddit, DIGEST achieves the best validation F1 score over training time of all the three frameworks. For dataset OGB-Arxiv, the performance of DIGEST is slightly worse than DGL but still outperforms LLCG. Specifically, LLCG’s training curves are not stable and fluctuate dramatically for both GCN and GAT on Reddit. This is because Reddit is much denser compared to other datasets, and in this case, the sampling process of the global server correction in LLCG has difficulty capturing all the information loss due to the cut-edges. Unlike LLCG, DIGEST’s training curves are much smoother not only for GCN training but also for GAT training.

B.2 Efficiency of Embedding Communication

In this section, we show the communication efficiency in terms of the ratio of the communication time to the overall training time per epoch. Since it is non-trivial to obtain this statistics from DGL, we only report the results of DIGEST. We can see from Figure 7 for smaller datasets OGB-Arxiv, Flickr and Reddit, the communication time including both pull and push operations takes less than 8% of the epoch time. Even for larger dataset OGB-Products that contains millions of nodes, the ratio is still low, only 12.96%, again demonstrating the communication efficiency of DIGEST.

C Theoretical Proof

In this section, we provide the formal proof for all the theories presented in the main paper.
Figure 6: Performance comparison of different distributed GAT training methods on four benchmark datasets. The top three subfigures show the training loss during the whole training process; the bottom three subfigures show the global validation F1 scores during the whole training process.

Figure 7: The ratio of average communication time to the average duration per epoch when training a GCN over the four datasets.

C.1 Proof of Section 4.1

We first introduce a useful lemma from [10].

**Lemma 2.** Given a $L$-layer GNN $f_{W}$ with $r_{1}$-Lipschitz smooth $\Phi$ and $r_{2}$-Lipschitz smooth $\Psi$. If $\forall \ v \in V$ and $\forall \ \ell \in \{1, 2, \ldots, L-1\}$ we have $\|h^{(\ell-1)}_{v} - h^{(\ell-1)}_{v}\| \leq \mu$ and $\|h^{(\ell-1)}_{v} - h^{(\ell-1)}_{v}\| \leq \epsilon$, then the embedding from the last layer (i.e., final output) of the GNN is bounded by

$$\|h^{(L+1)}_{v} - h^{*^{(L+1)}_{v}}\| \leq \mu r_{2} + (\mu + \epsilon) r_{1} r_{2} \mathcal{N}(v) \tag{9}$$

where $h^{*^{(\ell)}_{v}}$ denotes the exact output from the $\ell$-th layer of GNN without any staleness.

Now we give the proof for our Theorem [1] as follow:
Theorem 4 (Formal version of Theorem 1). Given a $L$-layer GNN $f_{W}$ with $r_1$-Lipschitz smooth $\Phi$ and $r_2$-Lipschitz smooth $\Psi$. If $\forall \ v \in V \ and \ \forall \ \ell \in \{1, 2, \cdots, L - 1\}$ we have $\|h_v^{(\ell)} - h_w^{(\ell)}\| \leq \epsilon^{(\ell)}$, then the embedding from the last layer (i.e., final output) of the GNN is bounded by

$$\|h_v^{(L)} - h_w^{(L)}\| \leq \sum_{\ell=1}^{L-1} \epsilon^{(\ell)} r_1^{L-\ell} r_2^{L-\ell} |\mathcal{N}(v)|^{L-\ell}$$

(10)

where $h_v^{(L)}$ denotes the exact output from the last layer of GNN without any staleness.

Proof. We prove the result by recursion under the distributed setting. Given any node $v \in V_m$, where $V_m$ is the nodes of $m$-th subgraph. Note for the first layer, i.e., $\ell = 1$, there is no approximation error since the input of the first layer is exactly the original input node feature matrix, so $\mu^{(1)} = \|h_v^{(0)} - h_w^{(0)}\| = 0$ and $\|h_v^{(1)} - h_w^{(1)}\| = 0$.

Based on Lemma 2 we know that, if $\|h_v^{(1)} - h_w^{(1)}\| \leq \epsilon^{(1)}$, we know that

$$\|h_v^{(2)} - h_w^{(2)}\| \leq \epsilon^{(1)} r_1 r_2 |\mathcal{N}(v)|$$

(11)

and we denote the term on left hand side above as $\mu^{(2)}$. And by recursively plugging

$$\mu^{(\ell+1)} = \mu^{(\ell)} r_2 + (\mu^{(\ell)} + \epsilon^{(\ell)}) r_1 r_2 |\mathcal{N}(v)|$$

(12)

into

$$\|h_v^{(L)} - h_w^{(L)}\| \leq \mu^{(L-1)} r_2 + (\mu^{(L-1)} + \epsilon^{(L-1)}) r_1 r_2 |\mathcal{N}(v)|$$

(13)

finishes the proof. \qed

Based on Theorem 4 we can quickly prove our Corollary 3 as follow:

Corollary 2 (Formal version of Corollary 1). Given a $L$-layer GNN $f_{W}$ with $r_1$-Lipschitz smooth $\Phi$ and $r_2$-Lipschitz smooth $\Psi$. If $\forall \ v \in V \ and \ \forall \ \ell \in \{1, 2, \cdots, L - 1\}$ we have $\|h_v^{(\ell)} - h_w^{(\ell)}\| \leq \epsilon^{(\ell)}$, then the gradients of GNN can be upper bounded by

$$\|\nabla_w \mathcal{L}_{m}^{Local}(h_v^{(L)}) - \nabla_w \mathcal{L}_{m}^{Local}(h_w^{(L)})\| \leq \tau \|h_v^{(L)} - h_w^{(L)}\|$$

$$\leq \tau \cdot \sum_{\ell=1}^{L-1} \epsilon^{(\ell)} r_1^{L-\ell} r_2^{L-\ell} |\mathcal{N}(v)|^{L-\ell}, \ \forall \ m \in [M]$$

(14)

if node $v$ is within the $m$-th subgraph and the loss function $\mathcal{L}_{m}^{Local}$ has $\tau$-Lipschitz smooth gradient w.r.t $W$.

Proof. Notice that $\forall \ m = 1, 2, \cdots, M$, each local loss function $\mathcal{L}_{m}^{Local}$ has $\tau$-Lipschitz smooth gradient, i.e., $\forall \ x$ and $y \in \mathbb{R}^{d_L}$,

$$\|\nabla_w \mathcal{L}_{m}^{Local}(x) - \nabla_w \mathcal{L}_{m}^{Local}(y)\| \leq \tau \|x - y\|$$

(15)

where $d_L$ denotes the dimension of the node embedding of the last layer. Plug in $h_v^{(L)}$ and $h_w^{(L)}$ and apply the conclusion of Theorem 4 finishes the proof. \qed

C.2 Proof of Section 4.2

Lemma 3 (Formal version of Lemma 1). Given countable embedding set $\{h_v^{(\ell-1)} : v \in V\}$. If $\|h_v^{(\ell-1)} - h_w^{(\ell-1)}\| > 2(\mu + \epsilon)$, $\forall \ v \neq w \in V$. If $\|h_v^{(\ell-1)} - h_w^{(\ell-1)}\| \leq \mu$ and $\|h_v^{(\ell-1)} - h_w^{(\ell-1)}\| \leq \epsilon$, then $\exists \ \Phi$ and $\Psi$, s.t.

$$\|f^{(\ell)}(h_v^{(\ell-1)}) - f^{(\ell)}(h_w^{(\ell-1)})\| \leq \mu + \epsilon$$

(16)

and

$$\|f^{(\ell)}(h_v^{(\ell-1)}) - f^{(\ell)}(h_w^{(\ell-1)})\| \geq 2(\mu + \epsilon + \lambda)$$

(17)

$\forall \ v \neq w \in V$ and $\lambda > 0$. $f$ is the GNN composed from $\Phi$ and $\Psi$. Parameters are omitted here for simpler notation.

Proof. Following [10], we first define a function $\xi : \mathbb{R}^d \rightarrow \mathbb{R}^d$ as the Voronoi tessellation based on the exact embedding $\{h_v^{(\ell-1)} : v \in V\}$, which can be defined as

$$\xi(x) = h_v^{(\ell-1)}, \ \text{if} \ \|x - h_v^{(\ell-1)}\| \leq \|x - h_w^{(\ell-1)}\|, \ \forall \ v \neq w \in V$$

(18)
According to [27], we know that there exist aggregation function $\Phi^{(\ell)}$ and updating function $\Psi^{(\ell)}$ such that the corresponding GNN layer $f^{(\ell)}$ is injective for any countable sets, which means
\[
\forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^d, \quad \text{if} \quad f^{(\ell)}(\mathbf{a}) = f^{(\ell)}(\mathbf{b}), \quad \text{then} \quad \mathbf{a} = \mathbf{b}
\]  \hfill (19)
which follows
\[
\|f^{(\ell)}(\xi(h_v^{(\ell-1)})) - f^{(\ell)}(\xi(h_v^{*(\ell-1)}))\| = 0 \leq \mu
\]  \hfill (20)
Since $f^{(\ell)}$ is injective and the embedding set is countable, there must exist a constant $\gamma > 0$ such that
\[
\|f^{(\ell)}(\xi(h_v^{(\ell-1)})) - f^{(\ell)}(\xi(h_v^{*(\ell-1)}))\| > \gamma
\]  \hfill (21)
\forall v, w \in V. Furthermore, due to the homogeneity of norm, i.e.,
\[
\|\alpha \cdot \mathbf{x}\| = |\alpha| \cdot \|\mathbf{x}\| \quad \forall \alpha \in \mathbb{R}
\]  \hfill (22)
we can finally find $\alpha > 0$ such that
\[
\|\alpha f^{(\ell)}(\xi(h_v^{(\ell-1)})) - f^{(\ell)}(\xi(h_v^{*(\ell-1)}))\| > \alpha \cdot \gamma \geq 2(\mu + \epsilon + \lambda), \quad \forall \lambda > 0
\]  \hfill (23)

**Theorem 5** (Formal version of Theorem 2). *Given a $L$-layer GNN $f$ which can be decomposed into $\Phi$ and $\Psi$ that satisfy the conditions in Lemma 7 then
\[
\exists \xi : \mathbb{R}^{dL} \to \Sigma, \quad \text{s.t.} \quad \xi(h_v^{(L)}) = c_v^{(L)}, \quad \forall v \in V
\]  \hfill (24)
where $c_v^{(L)}$ denotes a node’s coloring after $L$ rounds of refinement and $d_L$ is the output layer width.*

**Proof.** Again, we define the function $\xi$ similar to that in Lemma 3 but this time for the output layer. The general idea is, by applying Lemma 3 repeatedly, we will find that $\|h_v^{(L)} - h_v^{*(L)}\| \leq \sum_{\ell=1}^{L-1} \epsilon^{(\ell)} = \delta^{(L)}$ and meanwhile $\|h_v^{(L)} - h_w^{(L)}\| > \sum_{\ell=1}^{L-1} 2\epsilon^{(\ell)}, \forall v \neq w \in V$, which finishes the proof. Please refer to Theorem 5 in [10] for how to apply Lemma 3 in more detail.

### C.3 Proof of Section 4.3

In this section, we prove the convergence of DIGEST. The general idea of our proof takes 3 steps: First, we prove that under mild assumptions the stepwise change of intermediate variables such as the node embeddings and the gradients are bounded by a constant, which is proportional to the learning rate $\eta$. Second, we show that the approximation error between DIGEST’s local gradient and the exact local gradient without staleness can be further upper-bounded by a constant, which is also proportional to the learning rate $\eta$. Third, we aggregate the approximation error for each local gradient and prove that the error for the global gradient is finally bounded by $\eta E/\sqrt{M}$, where $M$ is the number of subgraphs and $E$ is a constant.

**Preliminary.** We consider GCN in our proof without loss of generality. We denote the input graph as $G = (V, E)$, $L$-layer GNN as $f$, feature matrix as $X$, weight matrix as $W$. The forward propagation of one layer of GCN is
\[
Z^{(\ell+1)} = PH^{(\ell)}W^{(\ell)}, \quad H^{(\ell+1)} = \sigma(Z^{(\ell)})
\]  \hfill (25)
where $\ell$ is the layer index, $\sigma$ is the activation function, and $P$ is the propagation matrix following the definition of GCN [16]. Notice $H^{(0)} = X$. We can further define the $(\ell + 1)$-th layer of GCN as:
\[
f^{(\ell+1)}(H^{(\ell)}, W^{(\ell)}) := \sigma(PH^{(\ell)}W^{(\ell)})
\]  \hfill (26)
The backward propagation of GCN can be expressed as follow:
\[
G_H^{(\ell)} = \nabla_H f^{(\ell+1)}(H^{(\ell)}, W^{(\ell)}, G_H^{(\ell+1)}) := P^T L_D^{(\ell+1)}(W^{(\ell+1)})^T
\]  \hfill (27)
\[
G_W^{(\ell+1)} = \nabla_W f^{(\ell+1)}(H^{(\ell+1)}, W^{(\ell)}, G_H^{(\ell+1)}) := (PH^{(\ell)})^T L_D^{(\ell+1)}
\]  \hfill (28)
where
\[
D^{(\ell+1)} = G_H^{(\ell)} \circ \sigma'(PH^{(\ell)}W^{(\ell+1)})
\]  \hfill (29)
and $\circ$ represents the Hadamard product.
As mentioned at the very beginning of this section, our proof of convergence can be regarded as 3 steps. The first step, where we use \( \tilde{\forall} \) Assumption 3.

The activation function is not part of the input since it is the historical results from the previous iteration, i.e., it can be changed. Notice that our proof is under the distributed setting which differentiates us with other works. We summarize are standard ones that are also used in [4, 25].

For DIGEST, the forward propagation of a single layer of GCN can be expressed as

\[
\begin{align*}
\tilde{Z}_m^{(t, \ell+1)} &= P_{in} \tilde{H}_m^{(t, \ell)} \tilde{W}_m^{(t, \ell)} + P_{out} \tilde{H}_m^{(t-1, \ell)} \tilde{W}_m^{(t, \ell)} \\
\tilde{H}_m^{(t+1)} &= \sigma(\tilde{Z}_m^{(t, \ell)})
\end{align*}
\]

where we use \( \tilde{H} \) to differentiate with the counterpart without staleness, i.e., \( H \) (same for other variables). \( t \) is the training iteration index. Similarly, we can define each layer as a single function

\[
\tilde{J}_m^{(t, \ell+1)}(\tilde{H}_m^{(t, \ell)}, \tilde{W}_m^{(t, \ell)}) := \sigma(P_{in} \tilde{H}_m^{(t, \ell)} \tilde{W}_m^{(t, \ell)} + P_{out} \tilde{H}_m^{(t-1, \ell)} \tilde{W}_m^{(t, \ell)})
\]

Note that \( \tilde{H}_m^{(t-1, \ell-1)} \) is not part of the input since it is the historical results from the previous iteration, i.e., it can be regarded as a constant in the current iteration.

Now we can give the definition of back-propagation in DIGEST:

\[
\begin{align*}
\tilde{G}_{H,m}^{(t, \ell)} &= \nabla_H \tilde{J}_m^{(t, \ell+1)}(\tilde{H}_m^{(t, \ell)}, \tilde{W}_m^{(t, \ell)}, \tilde{G}_{H,m}^{(t, \ell+1)}) \\
&:= P_{in}^{T} \tilde{D}_m^{(t, \ell+1)}(\tilde{W}_m^{(t, \ell+1)})^{T} + P_{out}^{T} \tilde{D}_m^{(t-1, \ell+1)}(\tilde{W}_m^{(t, \ell+1)})^{T} \\
\tilde{G}_{W,m}^{(t, \ell+1)} &= \nabla_W \tilde{J}_m^{(t, \ell+1)}(\tilde{H}_m^{(t, \ell+1)}, \tilde{W}_m^{(t, \ell)}, \tilde{G}_{H,m}^{(t, \ell+1)}) \\
&:= (P_{in} \tilde{H}_m^{(t, \ell)} + P_{out} \tilde{H}_m^{(t-1, \ell-1)})^{T} \tilde{D}_m^{(t, \ell+1)}
\end{align*}
\]

where

\[
\tilde{D}_m^{(t, \ell+1)} = G_{H,m}^{(t, \ell)} \circ \sigma'(P_{in} \tilde{H}_m^{(t, \ell)} \tilde{W}_m^{(t, \ell)} + P_{out} \tilde{H}_m^{(t-1, \ell-1)} \tilde{W}_m^{(t, \ell)})
\]

In our proof, we use \( \mathcal{L}(W^{(t)}) \) to denote the global loss with GCN parameter \( W \) after \( t \) iterations, and use \( \hat{L}_m(W_m^{(t)}) \) to denotes the local loss for the \( m \)-th subgraph with model parameter \( W_m^{(t)} \) after \( t \) iterations computed by DIGEST.

**Step 1.** We first make the following assumptions about the GCN model and the original input graph. These assumptions are standard ones that are also used in [4, 25].

**Assumption 1.** The loss function \( \text{Loss}(\cdot, \cdot) \) is \( C_{\text{Loss}} \)-Lipschitz continuous and \( L_{\text{Loss}} \)-Lipschitz smooth with respect to the last layer’s node embedding, i.e.,

\[
||\text{Loss}(h_v^{(L)}, y_v) - \text{Loss}(h_w^{(L)}, y_v)|| \leq C_{\text{Loss}}||h_v^{(L)} - h_w^{(L)}||_2
\]

and

\[
||\nabla\text{Loss}(h_v^{(L)}, y_v) - \nabla\text{Loss}(h_w^{(L)}, y_v)||_2 \leq L_{\text{Loss}}||h_v^{(L)} - h_w^{(L)}||_2
\]

**Assumption 2.** The activation function \( \sigma(\cdot) \) is \( C_{\sigma} \)-Lipschitz continuous and \( L_{\sigma} \)-Lipschitz smooth, i.e.

\[
||\sigma(Z_1^{(L)}) - \sigma(Z_2^{(L)})||_2 \leq C_{\sigma}||Z_1^{(L)} - Z_2^{(L)}||_2 \quad \text{and} \quad ||\sigma'(Z_1^{(L)}) - \sigma'(Z_2^{(L)})||_2 \leq L_{\sigma}||Z_1^{(L)} - Z_2^{(L)}||_2
\]

**Assumption 3.** \( \forall \ell \) that \( \ell = 1, 2, \cdots, L \), we have

\[
||W^{(\ell)}||_F \leq K_W, \quad ||P^{(\ell)}||_F \leq K_W, \quad ||X^{(\ell)}||_F \leq K_X.
\]

As mentioned at the very beginning of this section, our proof of convergence can be regarded as 3 steps. The first step, as will be introduced here, is to show that all intermediate variables as well as the gradients have a bounded stepwise change. Notice that our proof is under the distributed setting which differentiates us with other works. We summarize the results in the following lemma.
Lemma 4. Under the assumptions defined earlier, \( \forall m \) that \( m = 1,2,\ldots,M \), and \( \forall \ell \) that \( \ell = 1,2,\ldots,L \), there exist constants \( K_{\nabla H} > 0 \) and \( K_D > 0 \), such that
\[
\| \tilde{G}_{H,m}^{\ell}(t-1) - \tilde{G}_{H,m}^{\ell}(t) \|_F \leq K_{\nabla H} \quad \text{and} \quad \| \tilde{D}_{m}^{\ell}(t-1) - \tilde{D}_{m}^{\ell}(t) \|_F \leq K_D
\]  
(39)

Proof. The general idea here is, by first noticing that the stepwise changes of the weight matrices, output matrices and the embedding matrices are upper bounded, we can use induction to derive the conclusion easily. According to Corollary A.2 and Lemma A.3 in [25], we know that the stepwise change of \( \tilde{W}_{m}^{\ell}(t), \tilde{Z}_{m}^{\ell}(t) \) and \( \tilde{H}_{m}^{\ell}(t) \) are all upper bounded by some constants. By mathematical induction similar to Lemma A.4 in [25] finishes the proof. 

Step 2. Our second step is to find the upper bounds for the approximation error of the gradients. In other words, we want to upper-bound the difference between the gradients calculated by DIGEST and the counterpart with full-graph awareness (i.e., no dropped edges, or ground truth). Note that we compare both methods with the same parameters. 

Lemma 5. Under the assumptions defined earlier, \( \forall m \) that \( m = 1,2,\ldots,M \), and \( \forall \ell \) that \( \ell = 1,2,\ldots,L \), there exist constants \( U_H > 0 \), \( U_{\nabla H} > 0 \) and \( U_D > 0 \), such that
\[
\| \tilde{H}_{m}^{\ell}(t) - H_{m}^{\ell}(t) \|_F \leq U_H
\]
\[
\| \tilde{G}_{H,m}^{\ell}(t) - G_{H,m}^{\ell}(t) \|_F \leq U_{\nabla H}
\]
\[
\| \tilde{D}_{m}^{\ell}(t) - D_{m}^{\ell}(t) \|_F \leq U_D
\]  
(40)

Proof. Note that the variable of \( H_{m}^{\ell}(t) \) is essentially the same as \( H^{\ell}(t) \) (vanilla GCN) except for the last layer, because in \( H_{m}^{\ell}(t) \) we consider mini-batch training so only those nodes inside \( G_{m} \) are used to calculate the loss. With such awareness, the proof can be done by directly following Lemma A.6 and A.7 in [25].

Lemma 6. Under the assumptions defined earlier, \( \forall m \) that \( m = 1,2,\ldots,M \), and \( \forall \ell \) that \( \ell = 1,2,\ldots,L \), there exist constants \( U_{\nabla W} > 0 \), such that 
\[
\| \tilde{G}_{W,m}^{\ell}(t) - G_{W,m}^{\ell}(t) \|_F \leq U_{\nabla W}
\]  
(41)

Proof. Note that 
\[
\| \tilde{G}_{W,m}^{\ell}(t) - G_{W,m}^{\ell}(t) \|_F = \|(P_m \tilde{H}_{m}^{\ell,t-1}(t) + P_{out} \tilde{H}_{m}^{\ell,t-1}(t)) \tilde{D}_{m}^{\ell}(t) - (PH_{m}^{\ell})^T D_{m}^{\ell}(t) \|_F
\]
\[
= \|(P_m \tilde{H}_{m}^{\ell,t-1}(t) + P_{out} \tilde{H}_{m}^{\ell,t-1}(t)) \tilde{D}_{m}^{\ell}(t) - (PH_{m}^{\ell})^T D_{m}^{\ell}(t)
\]
\[
+ (PH_{m}^{\ell})^T D_{m}^{\ell}(t) - (PH_{m}^{\ell})^T D_{m}^{\ell}(t) \|_F
\]
\[
\leq \|(P_m \tilde{H}_{m}^{\ell,t-1}(t) + P_{out} \tilde{H}_{m}^{\ell,t-1}(t)) \tilde{D}_{m}^{\ell}(t) - (PH_{m}^{\ell})^T D_{m}^{\ell}(t) \|_F
\]
\[
+ \|(PH_{m}^{\ell})^T D_{m}^{\ell}(t) - (PH_{m}^{\ell})^T D_{m}^{\ell}(t) \|_F
\]  
(42)

On the one hand, 
\[
\| (P_m \tilde{H}_{m}^{\ell,t-1}(t) + P_{out} \tilde{H}_{m}^{\ell,t-1}(t)) \tilde{D}_{m}^{\ell}(t) - (PH_{m}^{\ell})^T D_{m}^{\ell}(t) \|_F
\]
\[
= \| (P_m \tilde{H}_{m}^{\ell,t-1}(t) + P_{out} \tilde{H}_{m}^{\ell,t-1}(t) - P_{out} \tilde{H}_{m}^{\ell,t-1}(t) - P_{out} \tilde{H}_{m}^{\ell,t-1}(t)) \tilde{D}_{m}^{\ell}(t)
\]
\[
- (PH_{m}^{\ell})^T D_{m}^{\ell}(t) \|_F
\]
\[
\leq B_D \| (P_{out} \tilde{H}_{m}^{\ell,t-1}(t) - H_{m}^{\ell,t-1}(t)) \|_F
\]
\[
\leq (L_H + K_{\nabla H})C_D C_P
\]  
(43)

where \( C_D \) and \( C_P \) is defined as the upper bound of matrix norm for \( D \) matrix and propagation matrix, respectively. They are guaranteed to exist by directly following our three assumptions in Step 1. 

On the other hand, 
\[
\| (PH_{m}^{\ell})^T D_{m}^{\ell}(t) - (PH_{m}^{\ell})^T D_{m}^{\ell}(t) \|_F \leq U_D C_H C_P
\]  
(44)

Combine these two parts together and denote the entire term as a new constant \( U_{\nabla W} \) finishes the proof. 

Lemma 7. \( \| \nabla L_m(W_m) - \nabla \hat{L}_m(W_m) \|_2 \leq E_{local} \), where \( E_{local} = U_{\nabla W} \).

Proof. Summing up over each layer, i.e., \( \ell = 1 \) up to \( L \) on both sides of the previous lemma finishes the proof.
Further, by some simple manipulation we can reformulate the result in the previous lemma as follow:

**Corollary 3.** \( \| \nabla L_m(W_m) - \nabla \tilde{L}_m(W_m) \|_2 \leq \eta E' \), where \( E' \) follows Corollary A.10 in [25].

**Step 3.** Now we can introduce the proof of our Theorem 3. We consider a GCN with \( L \) layers that is \( L_f \)-Lipschitz smooth, i.e., \( \| \nabla L(W_1) - \nabla L(W_2) \|_2 \leq L_f \| W_1 - W_2 \|_2 \).

**Theorem 6** (Formal version of Theorem 3). There exists a constant \( E \) such that for any arbitrarily small constant \( \epsilon > 0 \), we can choose a learning rate \( \eta = \frac{\sqrt{M\epsilon}}{E} \) and number of training iterations \( T = (L(W^{(1)}) - L(W^*)) \frac{E}{\sqrt{M}} \epsilon^{-\frac{3}{2}} \), such that

\[
\frac{1}{T} \sum_{t=1}^{T} \| \nabla L(W^{(t)}) \|^2 \leq O\left( \frac{1}{T^2 M^2} \right)
\]

where \( W^{(t)} \) and \( W^* \) denotes the parameters at iteration \( t \) and the optimal one, respectively.

**Proof.** Beginning from the assumption of smoothness of loss function,

\[
\mathcal{L}(W^{(t+1)}) \leq \mathcal{L}(W^t) + \left< \nabla \mathcal{L}(W^t), W^{(t+1)} - W^t \right> + \frac{L_f}{2} \| W^{(t+1)} - W^t \|_2^2
\]

Recall that the update rule of DIGEST is

\[
W^{(t+1)} = W^t - \frac{\eta}{M} \sum_{m=1}^{M} \nabla \tilde{L}_m(W_m^{(t)})
\]

so we have

\[
\mathcal{L}(W^t) + \left< \nabla \mathcal{L}(W^t), W^{(t+1)} - W^t \right> + \frac{L_f}{2} \| W^{(t+1)} - W^t \|_2^2
\]

\[
= \mathcal{L}(W^t) - \eta \left< \nabla \mathcal{L}(W^t), \frac{1}{M} \sum_{m=1}^{M} \nabla \tilde{L}_m(W_m^{(t)}) \right> + \frac{\eta^2 L_f}{2} \left\| \frac{1}{M} \sum_{m=1}^{M} \nabla \tilde{L}_m(W_m^{(t)}) \right\|_2^2
\]

Denote \( \delta_m^{(t)} = \nabla \tilde{L}_m(W_m^{(t)}) - \nabla \tilde{L}_m(W_m^{(t)}) \), we have

\[
\mathcal{L}(W^{(t+1)}) \leq \mathcal{L}(W^t) - \eta \left< \nabla \mathcal{L}(W^t), \frac{1}{M} \sum_{m=1}^{M} \left( \nabla \tilde{L}_m(W_m^{(t)}) + \delta_m^{(t)} \right) \right>
\]

\[
+ \frac{\eta^2 L_f}{2} \left\| \frac{1}{M} \sum_{m=1}^{M} \left( \nabla \tilde{L}_m(W_m^{(t)}) + \delta_m^{(t)} \right) \right\|_2^2
\]

Without loss of generality, assume the original graph can be divided evenly into \( M \) subgraphs and denote \( N = |V| \) as the original graph size, i.e., \( N = M \cdot S \), where \( S \) is each subgraph size. Notice that

\[
\nabla \mathcal{L}(W^t) = \frac{1}{N} \sum_{i=1}^{N} \nabla \text{Loss}(f_i^{(L)}, y_i) = \frac{1}{M} \left\{ \sum_{m=1}^{M} \frac{1}{S} \sum_{i=1}^{S} \nabla \text{Loss}(f_{m,i}^{(L)}, y_{m,i}) \right\}
\]

which is essentially

\[
\nabla \mathcal{L}(W^t) = \frac{1}{M} \sum_{m=1}^{M} \nabla \mathcal{L}_m(W_m^{(t)})
\]

Plugging the equation above into Eq. 49 we have

\[
\mathcal{L}(W^{(t+1)}) \leq \mathcal{L}(W^t) - \frac{\eta}{2} \| \nabla \mathcal{L}(W^t) \|_2^2 + \frac{\eta^2 L_f}{2} \left\| \frac{1}{M} \sum_{m=1}^{M} \delta_m^{(t)} \right\|_2^2
\]

which after rearranging the terms leads to

\[
\| \nabla \mathcal{L}(W^t) \|_2^2 \leq \frac{2}{\eta} (\mathcal{L}(W^t) - \mathcal{L}(W^{(t+1)})) + \eta L_f \left\| \frac{1}{M} \sum_{m=1}^{M} \delta_m^{(t)} \right\|_2^2
\]
By taking $\eta < 1/L_f$, using Corollary $[3]$ and summing up the inequality above over all iterations, i.e., $t = 1, 2, \cdots, T$, we have

$$
\frac{1}{T} \sum_{t=1}^{T} \|\nabla \mathcal{L}(W(t))\|^2 \leq \frac{2}{\eta T} (\mathcal{L}(W^1) - \mathcal{L}(W^{T+1})) + \frac{\eta^2 E^2}{M} \\
\leq \frac{2}{\eta T} (\mathcal{L}(W^1) - \mathcal{L}(W^*) + \frac{\eta^2 E^2}{M} 
$$

(54)

where $W^*$ denotes the minima of the loss function and $E$ is a constant depends on $E'$. Finally, taking $\eta = \sqrt{\frac{M}{E}}$ and $T = (\mathcal{L}(W(1)) - \mathcal{L}(W^*)) \frac{E}{\sqrt{M}} \epsilon^{-2}$ finishes the proof. 

$\square$