# GraphFM: Improving Large-Scale GNN Training via Feature Momentum

Haiyang Yu*[1]  Limei Wang*[1]  Bokun Wang*[2]  Meng Liu[1]  Tianbao Yang[2]  Shuiwang Ji[1]

## Abstract

Training of graph neural networks (GNNs) for large-scale node classification is challenging. A key difficulty lies in obtaining accurate hidden node representations while avoiding the neighborhood explosion problem. Here, we propose a new technique, named feature momentum (FM), that uses a momentum step to incorporate historical embeddings when updating feature representations. We develop two specific algorithms, known as GraphFM-IB and GraphFM-OB, that consider in-batch and out-of-batch data, respectively. GraphFM-IB applies FM to in-batch sampled data, while GraphFM-OB applies FM to out-of-batch data that are 1-hop neighborhood of in-batch data. We provide a convergence analysis for GraphFM-IB and some theoretical insight for GraphFM-OB. Empirically, we observe that GraphFM-IB can effectively alleviate the neighborhood explosion problem of existing methods. In addition, GraphFM-OB achieves promising performance on multiple large-scale graph datasets.

## 1. Introduction

Graph neural networks (GNNs) achieve promising performance on many graph learning tasks, such as node classification (Kipf & Welling, 2017; Hamilton et al., 2017; Velickovic et al., 2018; Liu et al., 2020), graph classification (Xu et al., 2018; Gao & Ji, 2019; Gao et al., 2021), link prediction (Zhang & Chen, 2018; Cai & Ji, 2020), and molecular property prediction (Gilmer et al., 2017; Liu et al., 2022a). In general, GNNs learn from graphs by the popular message passing framework (Gilmer et al., 2017). Specifically, we usually perform a recursive aggregation scheme in which each node aggregates representa-

tions from all 1-hop neighbors. Various GNNs (Kipf & Welling, 2017; Velickovic et al., 2018; Xu et al., 2018; Li et al., 2019) mainly differ in the employed aggregation functions. Such recursive aggregation scheme has been shown to be effective for learning graph representations. However, it leads to the inherent neighborhood explosion problem (Hamilton et al., 2017), since the number of neighbors grows exponentially with the depth of GNNs.

Due to such inherent problem, we have difficulties to apply GNNs to large-scale graphs. Notably, many real-world graphs, such as citation networks, social networks, and co-purchasing networks, are large-scale graphs (Hu et al., 2021) with massive numbers of nodes and edges. Thus, it is hard to obtain the complete computational graph, which contains the exploded neighborhood, with limited GPU memory when training on large-scale graphs. To tackle this, efficient training algorithms (Huang et al., 2018; Gao et al., 2018; Bojchevski et al., 2020; Chen et al., 2020a; Huang et al., 2021; You et al., 2020; Li et al., 2021; Wan et al., 2022; Liu et al., 2022b) have been developed to update model parameters with reasonable computational complexity and memory consumption. They aim to obtain accurate hidden node representations and gradient estimations while avoiding the neighborhood explosion problem. The mainstream efficient training algorithms are sampling methods. They perform node-wise, layer-wise, or graph sampling to alleviate the neighborhood explosion problem. However, sampling incurs unavoidable errors in estimation on hidden node embedding and gradients. In section 3, we formulate GNNs as recursive nonlinear functions and show that the gradients of the sampling methods suffer from estimation error due to the nonlinearity. In addition, establishing a convergence guarantee of Adam-style algorithms is another challenge for sampling based methods (Cong et al., 2021).

In this work, we propose a novel technique, known as feature momentum (FM), to address these problems. FM applies a momentum step on historical node embeddings for estimating accurate hidden node representations. Based on FM, we develop two algorithms based on sub-sampled node estimation and pseudo-full neighborhood estimation, respectively. The first algorithm, known as GraphFM-IB, applies FM after sampling the in-batch nodes. GraphFM-IB samples the 1-hop neighbors of target nodes recursively

*Equal contribution [1]Department of Computer Science & Engineering, Texas A&M University, TX, USA [2]Department of Computer Science, The University of Iowa, IA, USA. Correspondence to: Shuiwang Ji <sji@tamu.edu>.

and then updates historical embeddings of them with aggregated embeddings from their sampled neighbors using a momentum step. The second algorithm, known as GraphFM-OB, uses cluster-based sampling to draw in-batch nodes and employs the FM to update historical embeddings of 1-hop out-of-batch nodes with the message passing from in-batch nodes. The operations of GraphFM-OB are given in Figure 1. GraphFM-IB was shown theoretically to converge to a stationary point with enough iterations and a constant batch size. GraphFM-OB is shown to have some theoretical insight of possibly alleviating the staleness problem of historical embeddings.

We perform extensive experiments to evaluate our methods on multiple large-scale graphs. Results show that GraphFM-IB outperforms GraphSAGE and achieves comparable results with other baselines. Importantly, when sampling only one neighbor, GraphFM-IB achieves similar performance as when large batch sizes are used, thus alleviating the neighborhood explosion problem. We also show that GraphFM-OB outperforms current baselines on various large-scale graph datasets.

## 2. Related Work

GNNs are powerful methods for learning graph representations (Gori et al., 2005; Scarselli et al., 2008). Commonly used GNNs include GCN (Kipf & Welling, 2017), GCNII (Chen et al., 2020b), and PNA (Corso et al., 2020). Due to the neighborhood explosion problem, full-batch training of GNNs on large-scale graphs incurs prohibitive GPU memory consumption. Therefore, it is practically desirable to develop efficient training strategies on large-scale graphs. Recently, several categories of sampling methods have been proposed to reduce the size of the computational graph during training, including node-wise, layer-wise, and graph sampling methods.

**Node-wise sampling methods** uniformly sample a fixed number of neighbors when performing recursive aggregation and usually learn model parameters using gradients on a batch of nodes instead of all nodes. Such training strategy is originally proposed in GraphSAGE (Hamilton et al., 2017). Then VR-GCN (Chen et al., 2018b) integrates historical embeddings with GraphSAGE to reduce the estimation variance. Moreover, it provides a convergence analysis, which demonstrates that VR-GCN can converge to a local optimum with infinite iterations. However, the analysis assumes the unbiasedness of stochastic gradients, which is usually unrealistic. In addition, although node-wise sampling methods reduce memory requirement to some extend, they still suffer from the neighborhood explosion problem.

**Layer-wise sampling methods** select a fixed number of nodes in each layer according to their defined sampling distribution. They overcome the neighborhood explosion problem because the number of neighbors grows only linearly with depths. FastGCN (Chen et al., 2018a) performs such sampling independently in each layer. In contrast, LADIES (Zou et al., 2019) moves one step forward to consider the dependency of sampled nodes between layers. They both provide the unbiasedness and variance analysis of node embedding for one-layer GNNs without non-linear function. MVS-GCN (Cong et al., 2020) further analyzes multi-layer GNNs with nonlinearity. It formulates the training of GNNs as a stochastic compositional optimization problem. Since the stochastic estimator of the gradient is biased, a large batch size is required to eliminate the bias and variance for the convergence guarantee. Layer-wise sampling methods can effectively alleviate the memory bottleneck. Nonetheless, layer-wise sampling methods have incurred computational overhead since we have to perform extensive sampling during training.

**Graph sampling methods** sample subgraphs or clusters to construct minibatches before training. Note that we only need to obtain such subgraphs or clusters once as a preprocessing step. Specifically, GraphSAINT (Zeng et al., 2019) samples nodes, edges or random walks to construct subgraphs. SHADOW (Zeng et al., 2021) samples subgraphs according to PageRank scores of local neighbors for each node. It builds subgraphs for each node and converts a node classification task into a graph classification task. ClusterGCN (Chiang et al., 2019) aims to reduce the effect of cutting graphs by splitting a graph into separate clusters with clustering algorithms. GNNAutoScale (Fey et al., 2021) proposes to incorporate historical embeddings of the dropped edges among the clusters, thereby obtaining more accurate full-batch neighborhood estimation.

Another orthogonal direction for large-scale graph training is precomputing methods (Wu et al., 2019; Frasca et al., 2020; Liu & Ji, 2022). They aggregate the multi-hop features for each node on the raw input features by precomputing and then feed them into subsequent models. Since the precomputing procedure does not involve any learnable parameters, the training of each node is independent. They are efficient but do not employ powerful GNNs with non-linearity.

## 3. The Proposed Feature Momentum Method

In this section, we present the proposed methods. We first introduce some notations, then present the motivation of the proposed algorithmic design, and then discuss two methods for improving GNN training.

**Notations.** Let $\mathcal{V} = \{1, \ldots, n\}$ denote a set of nodes with input features denoted by $\mathbf{x}_v \in \mathbb{R}^{d_0}$ for node $v$. For any node $v$, we denote by $\mathcal{N}_v$ the neighboring nodes of $v$ that

have connections with $v$. Let $h_v^k \in \mathbb{R}^{d_k}$ denote the feature vector of the $v$-th node at the $k$-th layer, where $h_v^0 = \mathbf{x}_v$ and $h_v^k, \forall k = 1, \ldots, K$ are recursively computed. Let $W^k$ denote the model parameters at the $k$-th layer. For simplicity, we consider supervised learning tasks, where at the output layer we optimize the following loss:

$$\min_{W_1, \ldots, W_{K+1}} F(W) = \sum_{v \in \mathcal{S}} \ell(W; h_v^K) \qquad (1)$$

where $\mathcal{S} \subset \mathcal{V}$ denotes a subset of nodes that of interest for supervised learning. Let $H^k = [h_1^k, \ldots, h_n^k] \in \mathbb{R}^{n \times d_k}$ denote concatenated representations of all nodes at $k$-th layer. We can write $F(W) = f_{K+1}(H^K)$ for some non-linear function $f_{K+1}$.

**Motivation of Algorithmic Design.** The feature representations of all nodes at each layer are recursively computed. We can express $H^k$ as

$$H^k = f_k(H^{k-1}),$$

where $f_k$ is a parameterized non-linear function. As a result, we can write the optimization problem as a multi-level nested function:

$$F(W) = f_{K+1} \circ f_K \circ \ldots \circ f_1(H^0). \qquad (2)$$

The two challenges in solving GNN optimization problem are (i) the representation of a node at a higher layer $h_v^k$ might depend on the representations of a large number nodes in the previous layer; (ii) not all nodes' representations can be re-computed at every iteration when the total number of nodes is large. To this end, the GNN training is usually companioned with sampling of nodes at each layer. However, using sub-sampled nodes to compute the feature representations at each layer might lead to a large optimization error when the sampled neighborhood of each node is not large enough. Historical embeddings of non-sampled neighboring nodes have been used to reduce this error. However, depending on the mini-batch size, the historical embeddings could be outdated and might also have a large error. To address these issues, we introduce node-wise momentum features. To motivate this idea, we consider the following two-level functions:

$$F(\mathbf{w}) = f_1 \circ f_2(\mathbf{w}).$$

We assume that $f_1, f_2$ and their gradients are expensive to evaluate but unbiased stochastic versions are readily computed. In particular, we let $f_1(\cdot; \xi)$ and $f_2(\cdot; \zeta)$ denote the stochastic versions of $f_1$ and $f_2$ depending on random variable $\xi$ and $\zeta$, such that

$$\mathbb{E}[f_1(\cdot; \xi)] = f_1(\cdot), \quad \mathbb{E}[\nabla f_1(\cdot; \xi)] = \nabla f_1(\cdot)$$
$$\mathbb{E}[f_2(\cdot; \zeta)] = f_2(\cdot), \quad \mathbb{E}[\nabla f_2(\cdot; \zeta)] = \nabla f_2(\cdot)$$

The gradient of $F(\mathbf{w})$ is given by $\nabla F(\mathbf{w}) = \nabla f_2(\mathbf{w})^\top \nabla f_1(f_2(\mathbf{w}))$. An unbiased estimation is give by $\nabla \widehat{F}(\mathbf{w}) = \nabla f_2(\mathbf{w}; \zeta)^\top \nabla f_1(f_2(\mathbf{w}); \xi)$. It is notable that if $f_2(\mathbf{w})$ inside $f_1$ is simply replaced by $f_2(\mathbf{w}; \zeta)$, it will lead to a biased estimator due to non-linearity of $f_1$ and hence suffers from a large estimator error. To address this issue, existing works for two-level stochastic compositional optimization problems have proposed to use the momentum estimator (i.e., moving average) of $f_2(\mathbf{w})$ by

$$\hat{f}_{2,t} = (1 - \beta)\hat{f}_{2,t-1} + \beta f_2(\mathbf{w}_t; \zeta). \qquad (3)$$

While this idea seems straightforward in light of multi-level stochastic compositional optimization, there are still several challenges to be addressed in the customization for GNN training: (i) the non-linear function $f_i$ in (2) does not have an unbiased estimator except for $f_{K+1}$; hence a different perspective of decomposition is required by viewing $f_k(H^{k-1}) = \sigma(\hat{f}_k(H^{k-1}))$, where $\sigma$ is a simple deterministic activation function and $\hat{f}_k(\cdot)$ is a linear function of input; (ii) we cannot obtain unbaised estimators for all nodes in $H^{k-1}$; hence coordinate-wise sampling needs to be considered for the analysis; (iii) if historical embeddings of out-of-batch nodes are used, which do not give unbiased estimator for each in-batch node, ad-hoc methods using the momentum averaging for out-of-batch nodes need to be developed; (iv) last but not least, how to provide theoretical analysis (e.g., convergence analysis) for the proposed methods by using Adam-style update, which is mostly used for GNN training. To the best of our knowledge, no convergence analysis of an Adam-style method has been given for multi-level compositional optimization.

In the following two subsections, we will address these issues. We consider two representative GNN training methods, i.e., sub-sampled neighborhood estimation and pseudo-full neighborhood estimation, where the former uses only sub-sampled neighbors for estimating the feature representations of in-batch nodes, and the latter uses all neighbors for estimating the feature representations of in-batch nodes except that for the out-of-batch neighbors historical embeddings are used. For sub-sampled neighborhood estimation, we develop a stochastic method by using feature momentum and provide a convergence analysis for the Adam-style method. For pseudo-full neighborhood estimation, we develop an ad-hoc method on top of a state-of-the-art method GNNAutoScale and also provide some theoretical analysis for the estimation error.

### 3.1. Feature Momentum for In-Batch Nodes

Let us consider the computation of one node's representation. A basic operator in GNN training is to compute new feature embeddings of each node from its neighborhood nodes using their lower level feature embeddings. This op-

## Algorithm 1 GraphFM-IB

**Require:** $\eta, \{\beta_{0,k}\}, \beta_1, \beta_2, \tilde{h}_v^{k,0} = 0, \forall v \in \mathcal{V}$
**Ensure:** $\mathbf{w}_T$
1: **for** $t = 1, ...T$ **do**
2:     **for** $k = 1, \ldots, K$ **do**
3:         Draw a batch of nodes $\mathcal{D}_k$
4:         **for** $v \in \mathcal{D}_k$ **do**
5:             Sample a neighborhood denoted by $\mathcal{B}_v$
6:             Compute $\tilde{h}_v^{k,t}$ according to (8)
7:             Compute $\hat{h}_v^{k,t}$ according to (9)
8:             Normalize $\hat{h}_v^{k,t}$ appropriately (in practice)
9:         **end for**
10:     **end for**
11:     Compute the stochastic gradient estimator $G_t$ by

$$G_t = \frac{1}{|\mathcal{D}_K|} \sum_{v \in \mathcal{D}_k} \nabla \ell(W_t; \hat{h}_v^{K,t})$$

12:     Compute $\mathbf{v}_1^{t+1} = (1 - \beta_1)\mathbf{v}_1^t + \beta_1 G_t$
13:     Compute $\mathbf{v}_2^{t+1} = (1 - \beta_2)\mathbf{v}_2^t + \beta_2 G_t^2$
14:     Update $W^{t+1} = W^t - \frac{\eta}{\sqrt{\mathbf{v}_2^{t+1}} + \epsilon_0} \mathbf{v}_1^{t+1}$
15: **end for**

---

erator can be expressed by the following two steps:

$$h_{\mathcal{N}_v}^k = \text{Aggregate}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}) \quad (4)$$

$$h_v^k = \sigma(W^k \cdot \text{Concat}(h_v^{k-1}, h_{\mathcal{N}_v}^k)), \quad (5)$$

where the first step aggregates the representations of the nodes in the immediate neighborhood of node $v$ into a single vector $h_{\mathcal{N}(v)}^k$, and the second step concatenates the node's current representation $h_v^{k-1}$, with the aggregated neighborhood vector and passes it through a non-linear layer with an activation function $\sigma(\cdot)$ and weights $W_k$. Of particular interest, we consider the mean aggregator.

$$\mathcal{A}(\{h_u^{k-1} : u \in \mathcal{N}_v \cup \{v\}\}) = \frac{1}{|\mathcal{N}_v| + 1} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{k-1}$$

$$h_v^k = \sigma\left(W^k \cdot \mathcal{A}(\{h_u^{k-1} : u \in \mathcal{N}_v \cup \{v\}\})\right),$$

where $\mathcal{A}(\cdot)$ denotes the mean operator. To tackle the first challenge that involves a large neighborhood size, we use feature momentum with stochastic sampling to estimate the aggregated feature vector (before taking sigmoid) at the $t$-th iteration. To this end, we let $\mathcal{B}_v \subset \mathcal{N}_v$ denote a sub-sampled neighborhood of node $v$, and let $\bar{\mathcal{B}}_v = \mathcal{B}_v \cup \{v\}$. Then, we estimate the aggregated feature vector by

$$\tilde{h}_v^{k,t} = (1 - \beta_{0,k})\tilde{h}_v^{k,t-1} + \beta_{0,k}\hat{\mathcal{A}}(\{\hat{h}_u^{k-1,t}, u \in \bar{\mathcal{B}}_v\}), \quad (6)$$

where $\beta_{0,k} \in (0,1)$ is a momentum parameter, and $\hat{\mathcal{A}}(\{h_u^{k-1}, u \in \bar{\mathcal{B}}_v\})$ denotes an unbiased estimator of

## Algorithm 2 GraphFM-OB

**Require:** $\eta, \{\beta_{0,k}\}, \beta_1, \beta_2, \tilde{h}_v^{k,0} = 0, \forall v \in \mathcal{V}$
**Ensure:** $\mathbf{w}_T$
1: **for** $t = 1, ...T$ **do**
2:     Draw a batch of nodes $\mathcal{D}_t$
3:     **for** $k = 1, \ldots, K$ **do**
4:         Compute $\tilde{h}_{v,t}^{k,t}$ for $v \notin \mathcal{D}_t$ according to (11)
5:         **for** $v \in \mathcal{D}_t$ **do**
6:             Compute $h_v^{k,t}$ according to (12)
7:             Normalize $h_v^{k,t}$ appropriately (in practice)
8:         **end for**
9:     **end for**
10:     Compute the stochastic gradient estimator $G_t$ by

$$G_t = \frac{1}{|\mathcal{D}_t|} \sum_{v \in \mathcal{D}_t} \nabla \ell(W_t; h_v^{K,t})$$

11:     Compute $\mathbf{v}_1^{t+1} = (1 - \beta_1)\mathbf{v}_1^t + \beta_1 G_t$
12:     Compute $\mathbf{v}_2^{t+1} = (1 - \beta_2)\mathbf{v}_2^t + \beta_2 G_t^2$
13:     Update $W^{t+1} = W^t - \frac{\eta}{\sqrt{\mathbf{v}_2^{t+1}} + \epsilon_0} \mathbf{v}_1^{t+1}$
14: **end for**

---

$\mathcal{A}(\{h_u^{k-1} : u \in \mathcal{N}_v \cup \{v\}\})$. We can compute it by

$$\hat{\mathcal{A}}(\{\hat{h}_u^{k-1,t}, u \in \bar{\mathcal{B}}_v\}) = \frac{1}{|\mathcal{N}_v + 1|} \hat{h}_v^{k-1,t} \quad (7)$$

$$+ \frac{|\mathcal{N}_v|}{|\mathcal{N}_v| + 1} \frac{1}{|\mathcal{B}_v|} \sum_{u \in \mathcal{B}_v} \hat{h}_u^{k-1,t}.$$

To tackle the second challenge that computing $\tilde{h}_v^k$ for all nodes $v \in \mathcal{V}$ is expensive, we only compute it for a sub-sampled set of nodes denoted by $\mathcal{D}_k$ (in-batch nodes), i.e.,

$$\tilde{h}_v^{k,t} = \quad (8)$$

$$\begin{cases} \tilde{h}_v^{k,t-1} & \text{if } v \notin \mathcal{D}_k \\ (1 - \beta_{0,k})\tilde{h}_v^{k,t-1} + \beta_{0,k}\hat{\mathcal{A}}(\{\hat{h}_u^{k-1,t}, u \in \bar{\mathcal{B}}_v\}) & \text{o.w.} \end{cases}$$

With these momentum features, we can update the next layer feature by

$$\hat{h}_v^{k,t} = \sigma(W_k^t \cdot \tilde{h}_v^{k,t}). \quad (9)$$

The above procedure will be repeated for $K$ times for computing the output feature representations $\hat{h}_v^{K,t}$ for sub-sampled $v \in \mathcal{D}_K$.

We present the detailed steps of the proposed method based on sub-sampled neighborhood estimation in Algorithm 1, to which we refer as GraphFM-IB. The model parameter $\mathbf{w}_{t+1}$ is updated by the Adam-style update.

**Convergence Analysis.** Next, we provide convergence analysis of GraphFM-IB. We show that GraphFM-IB converges to a stationary solution after a large number of iterations without using a large neighborhood size, which can
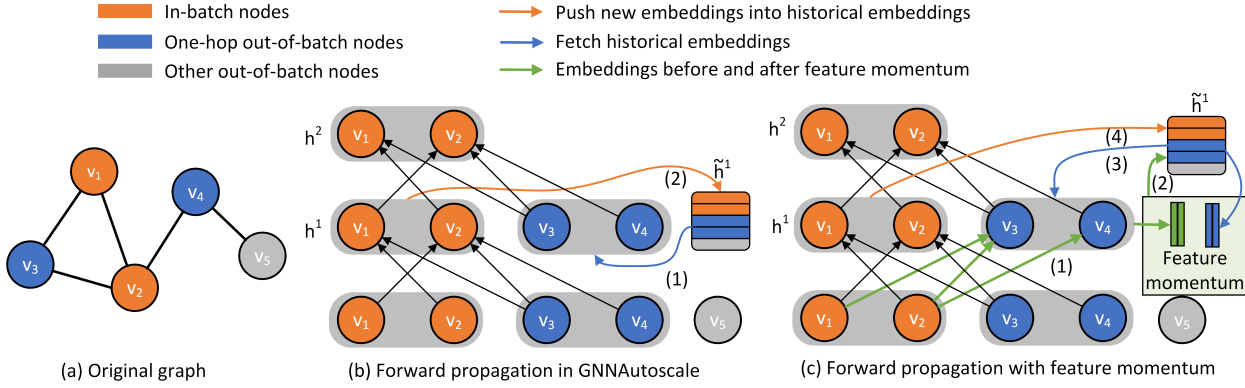
Figure 1: Comparsion of GrpahFM-OB with GNNAutoScale. (a) shows the original graph with in-batch nodes, one-hop out-of-batch nodes in blue and orange, respectively. (b) denotes the forward propagation in GNNAutoScale composed of two steps. The fist step fetches the historical embeddings for the one-hop out-of-batch nodes. Then it saves the in-batch nodes activation to their historical embeddings. Next, pseudo-full neighborhood propagation can be done to estimate the node embeddings of in-batch nodes in the next layer. (c) is the forward procedures in GraphFM-OB. It contains four steps. The first step is to calculate the message passing from the in-batch nodes to the out-batch nodes. Then we apply feature momentum to update the historical embeddings of the one-hop out-of-batch nodes, and save them into the corresponding historical embeddings. The third and last step is the same as the forward procedure in GNNAutoScale.

effectively avoid the neighbor explosion issue of existing node-wise sampling methods for large-scale GNN training. The detailed proof is provided in Appendix C.1.

**Theorem 1.** *Under proper conditions, with* $\eta = O(\epsilon^K), \beta_1 = O(\epsilon^K), 0 < \beta_2 < 1, \beta_{0,k} = O(\epsilon^{K-k}), T = O(\epsilon^{-(K+2)})$, *GraphFM-IB ensures to find an $\epsilon$-stationary solution such that* $\mathbb{E}[\|\nabla F(\mathbf{w}_\tau)\|] \leq \epsilon$ *for a randomly selected* $\tau \in \{1, \ldots, T\}$.

### 3.2. Feature Momentum for Out-of-Batch Nodes

In this subsection, we present a method based on GNNAutoScale by applying Feature Momentum to out-of-batch nodes. To this end, we first describe GNNAutoScale (Fey et al., 2021). It is based on pseudo-full neighborhood estimation that uses all neighbors for computing the new feature representation of an in-batch node. However, not all neighbors have updated their feature representations in the previous layer due to that some are not in the sampled batch. To address this issue, historical embeddings of those out-of-batch nodes are used. Let $\mathcal{D}_t$ denote the in-batch nodes. Then for each node $v \in \mathcal{D}_t$, we denote its neighborhood including itself by $\overline{\mathcal{N}_v} = \mathcal{N}_v \cup \{v\}$. We can decompose it into two subsets, $\mathcal{S}_v^t = \overline{\mathcal{N}_v} \cap \mathcal{D}_t$ and $\mathcal{O}_v^t = \overline{\mathcal{N}_v} \setminus \mathcal{S}_v^t$. In GNNAutoScale, the $k$-th layer embedding of node $v \in \mathcal{D}_t$ is computed as:

$$\hat{h}_v^{k,t} = W_k^t \cdot \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\} \cup \{\tilde{h}_u^{k-1,t} : u \in \mathcal{O}_v^t\})$$
$$h_v^{k,t} = \sigma(\hat{h}_v^{k,t}) \tag{10}$$

where $\tilde{h}_u^{k-1,t}$ denotes the fetched historical embedding of the out-batch node $u \in \mathcal{O}_v^t$. We denote by $\tau_u$ ($\tau_u < t$) the

last iteration before $t$ that $u$ is sampled, i.e., $u \in \mathcal{D}_{\tau_u}$. Due to the update rule of GNNAutoscale, it is worth noting that the historical embedding $\tilde{h}_u^{k-1}$ of $u$ in layer $k-1$ is not updated between iteration $\tau_u$ and $t$, i.e., $\tilde{h}_u^{k-1,t} = \ldots = \tilde{h}_u^{k-1,\tau_u} = h_u^{k-1,\tau_u}$. Depending on the batch size and the size of the graph, the last updated feature representation of node $u$ could happen a large number of iterations ago $\tau_u \ll t$, hence $\tilde{h}_u^{k-1,t} = h_u^{k-1,\tau_u}$ might have a large estimation error compared to $h_u^{k-1,t}$. In practice, cluster-based sampling has been shown to be helpful to reduce $t - \tau_u$ (Fey et al., 2021).

To further mitigate this issue, we notice that for those in out-of-batch (at layer $k$) such that their 1-hop neighbors are in the sampled batch (at layer $k-1$), we can use their in-batch neighbors to update their historical embeddings using moving average similar to that in (6). Then we update the historical embedding $\tilde{h}_v^{k,t}$ for $v \notin \mathcal{D}_t$ by

$$\tilde{h}_v^{k,t} = (1 - \beta_{0,k})\tilde{h}_v^{k,t-1} + \beta_{0,k}\sigma(W_k^t \cdot \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\})). \tag{11}$$

Indeed, we do not need to update $\tilde{h}_v^{k,t}$ for all $v \notin \mathcal{D}_t$. We only need to consider those such that $\mathcal{S}_v^t \neq \emptyset$, $v \notin \mathcal{D}_t$.

Then, we can update $h_v^{k,t}$ for $v \in \mathcal{D}_t$ by

$$\hat{h}_v^{k,t} = \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\} \cup \{\tilde{h}_u^{k-1,t} : u \in \mathcal{O}_v^t\})$$
$$h_v^{k,t} = \sigma(W_k^t \cdot \hat{h}_v^{k,t}). \tag{12}$$

We present a formal description of the proposed method GraphFM-OB in Algorithm 2. A computational graph of

Table 1: Statistics and properties of the datasets. The "m" denotes the multi-label classification task, and "s" denotes single label classification task.

| Dataset | # of nodes | # of edges | Avg. degree | # of features | # of classes | Train/Val/Test |
|---|---|---|---|---|---|---|
| Flickr | 89,250 | 899,756 | 10.0813 | 500 | 7(s) | 0.500/0.250/0.250 |
| Yelp | 716,847 | 6,997,410 | 9.7614 | 300 | 50(m) | 0.750/0.150/0.100 |
| Reddit | 232,965 | 11,606,919 | 49.8226 | 602 | 41(s) | 0.660/0.100/0.240 |
| ogbn-arxiv | 169,343 | 1,166,243 | 6.8869 | 128 | 40(s) | 0.537/0.176/0.287 |
| ogbn-products | 2,449,029 | 61,859,140 | 25.2586 | 100 | 47(s) | 0.100/0.020/0.880 |
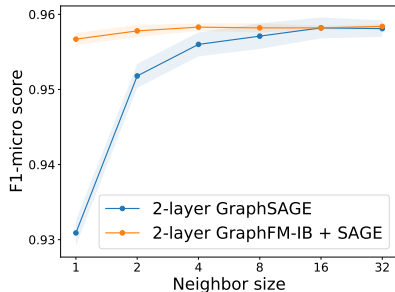


(a) Comparison between GraphSAGE and graphFM-IB + SAGE on Reddit.
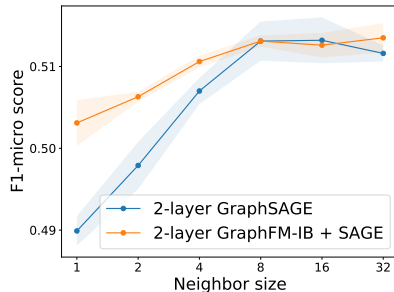
(b) Comparison between GraphSAGE and graphFM-IB + SAGE on Flickr.
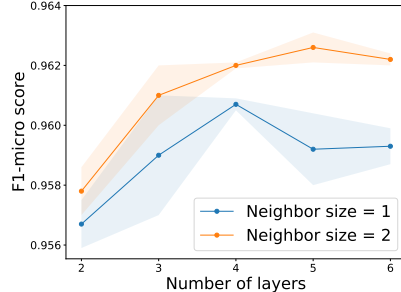
(c) Results for graphFM-IB + SAGE with different number of layers and neighbor sizes on Reddit.

Figure 2: Illustration of the difference between GraphSAGE and graphFM-IB + SAGE and the performance of graphFM-IB + SAGE with different number of layers. Neighbor size denotes the sampled neighbor size for each node at every layer.

GraphFM-OB compred with GNNAutoScale is shown in Figure 1. Some theoretical insight showing that the updated $h_v^{k,t}$ in (12) by GraphFM-OB could lead to smaller estimation error than the updated $h_v^{k,t}$ in (10) by GNNAutoScale can be found in Appendix B.

## 4. Experiments

**Datasets.** We evaluate our proposed algorithms GraphFM-IB and GraphFM-OB with extensive experiments on the node classification task on five large-scale graphs, including Flickr (Zeng et al., 2019), Yelp (Zeng et al., 2019), Reddit (Hamilton et al., 2017), ogbn-arxiv (Hu et al., 2021) and ogbn-products (Hu et al., 2021). They contains thousands or millions of nodes and edges, and we summarize the statistics of these datasets in Table 1. The detailed task description of these datasets can be found in Appendix A.1.

**Baselines.** We compare with the following five baselines: 1. VR-GCN (Chen et al., 2018b), 2. FastGCN (Chen et al., 2018a), 3. GraphSAINT (Zeng et al., 2019), 4. ClusterGCN (Chiang et al., 2019), 5. SIGN (Frasca et al., 2020). They cover different categories of efficient algorithms on large-scale graph training, including node-wise, layer-wise, graph sampling and precomptuing methods. Comparison and details about these baselines are provided in Appendix

A.2.

**Software and Hardware.** The implementation of our methods is based on the PyTorch (Paszke et al., 2019), and Pytorch_geometric (Fey & Lenssen, 2019). Our code is implemented in the DIG (Dive into Graphs) library (Liu et al., 2021), which is a turnkey library for graph deep learning research and publicly available[1]. In addition, we conduct our experiments on Nvidia GeForce RTX 2080 with 11GB memory, and Intel Xeon Gold 6248 CPU.

### 4.1. Feature Momentum for In-Batch Nodes

**Setup.** We first apply our proposed GraphFM-IB algorithm in the framework of GraphSAGE (Hamilton et al., 2017) and conduct experiments on five large-scale graph datasets. GraphSAGE is designed for large-scale graph learning and provides a general framework with neighbor sampling and aggregation. The aggregation is implemented in a graph convolution layer, called SAGEConv. The main idea of the neighbor sampling is to sample neighbors for each node to avoid the memory issue caused by considering full neighbors. In practice, the authors use a fixed-size, uniform sampling function to sample neighbors for each node in each layer. The fixed neighbor sizes can be different among lay-

---

[1]https://github.com/divelab/DIG/tree/dig/dig/lsgraph

Table 2: Comparison between our GraphFM-IB, GraphFM-OB and other baseline methods. The reported results of GraphFM-IB have averaged over 5 random runs. The experiments of GraphFM-OB follow the setting of GNNAutoScale to report the F1-micro scores with fixed random seeds for a fair comparison. The top performance scores are highlighted in **bold**. <u>Underline</u> indicates that our methods achieve better performance compared to the corresponding baselines without feature momentum.

| Backbones | Methods | Flickr | Reddit | Yelp | ogbn-arxiv | ogbn-products |
|---|---|---|---|---|---|---|
| | VR-GCN | 0.482 ± 0.003 | 0.964 ± 0.001 | 0.640 ± 0.002 | – | – |
| | FastGCN | 0.504 ± 0.001 | 0.924 ± 0.001 | 0.265 ± 0.053 | – | – |
| | GraphSAINT | 0.511 ± 0.001 | 0.966 ± 0.001 | 0.653 ± 0.003 | – | 0.791 ± 0.002 |
| | Cluster-GCN | 0.481 ± 0.005 | 0.954 ± 0.001 | 0.609 ± 0.005 | – | 0.790 ± 0.003 |
| | SIGN | 0.514 ± 0.001 | 0.968 ± 0.000 | 0.631 ± 0.003 | 0.720 ± 0.001 | 0.776 ± 0.001 |
| SAGE | GraphSAGE | 0.501 ± 0.013 | 0.953 ± 0.001 | 0.634 ± 0.006 | 0.715 ± 0.003 | 0.783 ± 0.002 |
| | GraphFM-IB | <u>0.513 ± 0.009</u> | <u>0.963 ± 0.005</u> | <u>0.641 ± 0.001</u> | 0.713 ± 0.002 | <u>0.792 ± 0.003</u> |
| GCN | GNNAutoScale | 0.5400 | 0.9545 | 0.6294 | 0.7168 | 0.7666 |
| | GraphFM-OB | <u>0.5446</u> | 0.9540 | – | <u>0.7181</u> | <u>0.7688</u> |
| GCNII | GNNAutoScale | 0.5620 | 0.9677 | 0.6514 | 0.7300 | 0.7724 |
| | GraphFM-OB | <u>0.5631</u> | <u>0.9680</u> | **<u>0.6529</u>** | **<u>0.7310</u>** | <u>0.7742</u> |
| PNA | GNNAutoScale | 0.5667 | **0.9717** | 0.6440 | 0.7250 | 0.7991 |
| | GraphFM-OB | **<u>0.5710</u>** | 0.9712 | <u>0.6450</u> | <u>0.7290</u> | **<u>0.8047</u>** |

ers and the authors use a 2-layer GraphSAGE with sampled neighbor sizes as 25 and 10. In the following part, we use 'GraphFM-IB + SAGE' to denote our GraphFM-IB method with SAGEConv. We explore the feature momentum hyper-parameter $\beta$ in the range from 0.1 to 0.9.

**Results.** We start by studying the effects of sampled neighbor size on the performance of GraphSAGE and GraphFM-IB + SAGE, and show that our proposed GraphFM-IB can achieve competitive performance with smaller sampled neighbor sizes. As shown in Fig. 2(a) and Fig. 2(b), the performance for 2-layer GraphSAGE is highly related to the sampled neighbor size. It requires to sample at least 8 neighbors per node to guarantee good performance. In contrast, GraphFM-IB + SAGE achieves good results with only 1 sampled neighbor. Thus we can reduce the sampled neighbor size while obtaining competitive performance by using feature momentum.

In the next step, we explore GNNs with more layers. Since GrpahFM-IB can perform well with small sampled neighbor sizes, we focus on the cases where the neighbor sizes are 1 or 2. As shown in Fig. 2(c), models with more layers outperform the 2-layer one. Note that using more than 5 layers may hurt the performance since deep model may need carefully designed architecture and training strategies, which is another research problem (Liu et al., 2020; Li et al., 2019).

To further evaluate the memory and time efficiency of the proposed GraphFM-IB, we compare GraphSAGE and

GraphFM-IB with various neighbor sizes in Table 3. We can observe that GraphFM-IB saves a lot of GPU memory and training time while achieving similar performance as GraphSAGE, especially when the GNNs have many layers. For example, to achieve similar performance with four-layer SAGE on Reddit, GraphSAGE with neighbor sizes $[25, 10, 10, 10]$ costs $10,100M$ and 53 seconds per epoch, while GraphFM-IB with neighbor sizes $[1, 1, 1, 1]$ costs $2,860M$ and 6.2 seconds per epoch. Note that $[1, 1, 1, 1]$ denotes sampling one neighbor at each layer in a 4-layer GNN. In this case, GraphFM-IB only samples one neighbor, thus alleviating the neighborhood explosion problem. Besides, the incremental GPU memory mainly caused by historical embeddings is acceptable. On Reddit, the incorporation of historical embedding costs 50M for two-layer SAGE and 160M for four-layer SAGE with neighbor sizes $[1, 1, 1, 1]$.

Finally, we explore the sampled neighbor size in the set {1,2,4,8} and the number of layers from 2 to 4. The testing F1-micro scores for GraphSAGE and the model using GraphFM-IB are summarized in Table 2. The results for GraphSAGE are taken from the referred papers and the OGB leaderboards. The results show that using our GraphFM-IB can consistently outperform GraphSAGE on all datasets, indicating that feature momentum is helpful for better feature estimation and improves performance for large graphs. Note that GraphSAGE with GraphFM-IB achieves almost the same results as GraphSAGE on ogb-arxiv since the result of GraphSAGE is full-batch training

Table 3: Comparison between with and without feature momentum of GraphSAGE in terms of **model performance, GPU memory consumption and running time per epoch** on Reddit and Flickr. Neighbor sizes are the list of number of neighbors sampled in each layer and the authors of GraphSAGE use the sampled neighbor sizes as 25 and 10.

| Methods | Neighbor sizes | Reddit | Flickr |
|---|---|---|---|
| GraphSAGE | 2 layer full-batch | OOM | 0.513/4,860M/1.7s |
| GraphSAGE | [25,10] | 0.957/3,080M/6.5s | 0.512/1,740M/1.6s |
| GraphSAGE | [1,1] | 0.931/2,250M/3.3s | 0.490/1,310M/1.2s |
| GraphFM-IB + SAGE | [1,1] | 0.957/2,300M/3.9s | 0.503/1,480M/1.4s |
| GraphSAGE | [4,4] | 0.955/2,320M/4.0s | 0.507/1,390M/1.3s |
| GraphFM-IB + SAGE | [4,4] | 0.958/2,450M/4.2s | 0.511/1,540M/1.5s |
| GraphSAGE | 4 layer full-batch | OOM | 0.514/11,000M/5.2s |
| GraphSAGE | [25,10,10,10] | 0.962/10,110M/53s | 0.514/6,480M/3.6s |
| GraphSAGE | [1,1,1,1] | 0.951/2,700M/5.2s | 0.502/1,360M/1.7s |
| GraphFM-IB + SAGE | [1,1,1,1] | 0.962/2,860M/6.2s | 0.513/1,700M/2.0s |
| GraphSAGE | [2,2,2,2] | 0.958/2,870M/5.8s | 0.509/1,470M/1.8s |
| GraphFM-IB + SAGE | [2,2,2,2] | 0.963/3,130M/7.5s | 0.513/1,900M/2.4s |

with the GraphSAGE convolution layer provided by the OGB team. We list the number here to keep consistent with the value in OGB leaderboards and the GraphSAGE result with neighbor sizes as 25 and 10 is 0.704 ± 0.001, which is still worse than GraphFM-IB + SAGE.

### 4.2. Feature Momentum for Out-Batch Nodes

**Setup.** To evaluate our proposed GraphFM-OB algorithm, we apply it to three widely used GNN backbones, including **GCN** (Kipf & Welling, 2017), **GCNII** (Chen et al., 2020b) and **PNA** (Corso et al., 2020). Then we conduct experiments to evaluate the obatined models on the five large-scale graphs. We explore the feature momentum hyper-parameter $\beta$ in the range from 0.1 to 0.9. We select the learning rate from {0.01, 0.05, 0.001} and dropout from {0.0, 0.1, 0.3, 0.5}. Due to the over-fitting problem on the ogbn-products dataset, we set the edge drop (Rong et al., 2019) ratio at 0.8 during training for this particular dataset.

**Results.** The testing F1-micro scores are shown in Table 2.

It can be observed that GraphFM-OB methods with different GNN backbones outperform corresponding baselines on four datasets. In addtion, GraphFM-OB shows enhancement performance with GCNII on all five datasets. With PNA as backbone, GraphFM-OB achieves the state-of-the-art performance on the Flickr dataset. These results demonstrate that GraphFM-OB, with using feature momentum to alleviate the staleness of historical embeddings, can obtain more accurate node embedding estimation.

As an exception, the implementation of GNNAutoScale and GraphFM-OB on Yelp with GCN perform much worse than the reported score 0.6294. Therefore, we omit the this

result in Table 2.

**Measuring the staleness of historical embeddings.** In order to measure the staleness of the historical embeddings, we propose a new metric called staleness score. Intuitively, if the historical embeddings of nodes are bright new without any staleness, they are exact the same as the full-neighborhood propagation embeddings. The staleness of the historical embeddings is the reason for the biasedness estimation of the pseudo full-neighborhood embeddings. Here we define the staleness score of node $v$ at $k$-th layer as Euclidean distance of its historical embedding at layer $k$ and full-neighborhood propagation embedding at $k$-th layer. Formally,

$$S^k(v) = \|\bar{h}_v^k - \tilde{h}_v^k\|, \qquad (13)$$

where $\bar{h}_v^k$ is the full-neighborhood propagation embedding of node $v$ at $k$-th layer.

Based on the staleness score of node $v$ at layer $k$, we further propose the staleness score for the layer $k$ averaged over all the nodes as

$$S^k = \frac{1}{|V|} \sum_{v \in \mathcal{V}} S^k(v) \qquad (14)$$

Thus, the $S^k$ can reflect the staleness of historical embeddings of all the node at layer $k$. In this way, we can compare different methods with such metric to evaluate the staleness of their historical embeddings.

We select the PNA as the backbone with 3, 4, 3 layers on Flickr, ogbn-arxiv and Yelp, respectively. We split these datasets into 24, 40 and 40 clusters respectively, and the batch size is set to 1. For GraphFM-OB, we select feature momentum hyper-parameter $\beta$ as 0.5, 0.3 and 0.3 for

Table 4: The staleness scores of the historical embeddings

| Datasets | Layer | GNNAutoScale | GraphFM-OB |
|---|---|---|---|
| Flickr | 1 | 3.8929 | **3.2046** |
| | 2 | 3.2185 | **2.3873** |
| ogbn-arxiv | 1 | 8.2709 | **5.7088** |
| | 2 | 12.5646 | **12.0062** |
| | 3 | 2.0200 | **1.4884** |
| Yelp | 1 | **3.0186** | 3.2484 |
| | 2 | 4.4013 | **3.8328** |

these three datasets respectively. We calculate the staleness scores for the historical embeddings after the epoch that achieves the best evaluation result.

The staleness scores are shown in Table 4. We can obviously find that the GraphFM-OB achieves smaller staleness scores than GNNAutoScale in most cases, indicating that GraphFM-OB can alleviate the staleness problem.

## 5. Conclusion

To obtain accurate hidden node representations, we propose feature momentum (FM) to incorporate historical embeddings in an Adam-update style. Based on FM, we develop two algorithms, GraphFM-IB and GraphFM-OB, with convergence guarantee and some theoretical insight, respectively. Extensive experiments demonstrate that our proposed methods can effectively alleviate the neighborhood explosion and the staleness problems, while achieving promising results.

## Acknowledgments

## References

Balasubramanian, K., Ghadimi, S., and Nguyen, A. Stochastic multi-level composition optimization algorithms with level-independent convergence rates. *arXiv preprint arXiv:2008.10526*, 2020.

Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., Rózemberczki, B., Lukasik, M., and Günnemann, S. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2464–2473, 2020.

Cai, L. and Ji, S. A multi-scale approach for graph link

prediction. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pp. 3308–3315, 2020.

Chen, J., Ma, T., and Xiao, C. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018a. URL https://openreview.net/forum?id=rytstxWAW.

Chen, J., Zhu, J., and Song, L. Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 941–949, 2018b.

Chen, M., Wei, Z., Ding, B., Li, Y., Yuan, Y., Du, X., and Wen, J.-R. Scalable graph neural networks via bidirectional propagation. *Advances in neural information processing systems*, 33:14556–14566, 2020a.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 1725–1735. PMLR, 2020b.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., and Hsieh, C.-J. 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.

Cong, W., Forsati, R., Kandemir, M., and Mahdavi, M. 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.

Cong, W., Ramezani, M., and Mahdavi, M. On the importance of sampling in training GCNs: Convergence analysis and variance reduction. *arXiv preprint arXiv:2103.02696*, 2021. URL https://openreview.net/forum?id=Oq79NOiZB1H.

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

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Fey, M., Lenssen, J. E., Weichert, F., and Leskovec, J. GNNAutoScale: Scalable and expressive graph neural networks via historical embeddings. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 3294–3304. PMLR, 2021.

Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., and Monti, F. SIGN: Scalable inception graph neural networks. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020.

Gao, H. and Ji, S. Graph U-Nets. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 2083–2092. PMLR, 2019.

Gao, H., Wang, Z., and Ji, S. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1416–1424. ACM, 2018.

Gao, H., Liu, Y., and Ji, S. Topology-aware graph pooling networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(12):4512–4518, 2021.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Guo, Z., Xu, Y., Yin, W., Jin, R., and Yang, T. A novel convergence analysis for algorithms of the adam family. *arXiv preprint arXiv:2112.03459*, 2021.

Hamilton, W. L., Ying, R., and Leskovec, J. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., and Leskovec, J. OGB-LSC: A large-scale challenge for machine learning on graphs. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Huang, Q., He, H., Singh, A., Lim, S.-N., and Benson, A. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=8E1-f3VhX1o.

Huang, W., Zhang, T., Rong, Y., and Huang, J. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Li, G., Muller, M., Thabet, A., and Ghanem, B. Deep-GCNs: Can GCNs go as deep as CNNs? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9267–9276, 2019.

Li, G., Müller, M., Ghanem, B., and Koltun, V. Training graph neural networks with 1000 layers. In *International conference on machine learning*, pp. 6437–6449. PMLR, 2021.

Liu, M. and Ji, S. Neighbor2Seq: Deep learning on massive graphs by transforming neighbors to sequences. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pp. 55–63. SIAM, 2022.

Liu, M., Gao, H., and Ji, S. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 338–348, 2020.

Liu, M., Luo, Y., Wang, L., Xie, Y., Yuan, H., Gui, S., Yu, H., Xu, Z., Zhang, J., Liu, Y., Yan, K., Liu, H., Fu, C., Oztekin, B. M., Zhang, X., and Ji, S. DIG: A turnkey library for diving into graph deep learning research. *Journal of Machine Learning Research*, 22(240):1–9, 2021. URL http://jmlr.org/papers/v22/21-0343.html.

Liu, Y., Wang, L., Liu, M., Lin, Y., Zhang, X., Oztekin, B., and Ji, S. Spherical message passing for 3D molecular graphs. In *International Conference on Learning Representations*, 2022a. URL https://openreview.net/forum?id=givsRXsOt9r.

Liu, Z., Zhou, K., Yang, F., Li, L., Chen, R., and Hu, X. EXACT: Scalable graph neural networks training via extreme activation compression. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=vkaMaq95_rX.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Rong, Y., Huang, W., Xu, T., and Huang, J. DropEdge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Wan, C., Li, Y., Wolfe, C. R., Kyrillidis, A., Kim, N. S., and Lin, Y. PipeGCN: Efficient full-graph training of graph convolutional networks with pipelined feature communication. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=kSwqMH0zn1F.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

You, Y., Chen, T., Wang, Z., and Shen, Y. L2-GCN: Layer-wise and learned efficient training of graph convolutional networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2127–2135, 2020.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., and Prasanna, V. GraphSAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2019.

Zeng, H., Zhang, M., Xia, Y., Srivastava, A., Malevich, A., Kannan, R., Prasanna, V., Jin, L., and Chen, R. Decoupling the depth and scope of graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31:5165–5175, 2018.

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., and Gu, Q. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems*, 2019.

# Appendix

## A. Experiment Settings

### A.1. Dataset Descriptions

We compare the results on the following five large-scale datasets. They cover many real world tasks, which are summarized as follows. 1. classifying the image tags with image description and edges to images with same properties. (Flickr) 2. classifying user types with their reviews and edges to their friends. (Yelp) 3. classifying the community of the posts with the posts content and edges to the posts that have been commented by the same customer. (Reddit) 4. classifying the papers with the abstract average embedding features and edges to their citation papers. (ogbn-arxiv) 5. classifying categories of products on Amazon with the product description and edges to other products that are purchased together. (ogbn-products) We follow the official split of these datasets to conduct our experiments.

### A.2. Baseline Descriptions

We compare the results with five different baselines, including node-wise, layer-wise, subgraph sampling and precomputing methods. We summarize the baselines as below. **VR-GCN** (Chen et al., 2018b) is a node-wise sampling method. Compared to the GraphSAGE, it combines the historical embedding of one-hop neighbors and the estimated embedding of sampled nodes to reduce the variance of the unbiased estimated target node embedding. **FastGCN** (Chen et al., 2018a) is a layer-wise sampling method. For each layer, it will sample fixed number of nodes from the neighborhood union of target nodes in the next layer with importance sampling to reduce the variance of unbiased estimated target node embeddings. Since it will sample fixed number of nodes in each layer, the total number of sampled nodes will grow linearly. However, for node-wise sampling method, each node will sample fixed number of neighbors from the previous layer. Thus, it will lead to an exponentially growth of sampled nodes. **ClusterGCN** (Chiang et al., 2019) and **GraphSAINT** (Zeng et al., 2019) are subgraph-sampling methods. ClusterGCN will combine fixed number of clusters to form the subgraphs to train the model. GraphSAINT will take sampled edges or random walks to form subgraphs. Compared to the layer-wise sampling method, once the subgraph is provided, the nodes in each layer are the same as the input subgraph. Therefore, the sampling procedure can be done once before training, which can alleviate the overhead of sampling. **SIGN** (Frasca et al., 2020) is a precomputing method. It computes the L-hop aggregated features on the raw input features first, then directly feed them into MLPs to predict the labels. Because the precomputing procedure doesn't contain any parameter, the training of each node is independent.

## B. Theoretical Insight of GraphFM-OB

Consider one layer of the graph neural network. The $\underline{h_v^{k,t}}$ representation of a node $v$ in the last layer by full-neighbordhood forward propagation can be written as:

$$\underline{h_v^{k,t}} = \sigma(\underline{\hat{h}_v^{k,t}}), \quad \underline{\hat{h}_v^{k,t}} = W_k^t \cdot \mathcal{A}(\{h_u^{k-1,t} : u \in \overline{\mathcal{N}}_v\}).$$

Instead of full-neighborhood propagation on every nodes, the GNNAutoScale algorithm samples a batch of nodes $\mathcal{D}_t$ and approximate $\underline{h_v^{k,t}}$ by $h_v^{k,t}$, which is computed by incorporating the current embedding $h_u^{k-1,t}$ of in-batch node $u \in S_v^t$ and the historical embedding $\tilde{h}_u^{k-1,t}$ of out-batch node $u \in \mathcal{O}_v^t$ as follows

$$h_v^{k,t} = \sigma(\hat{h}_v^{k,t}), \quad \hat{h}_v^{k,t} = W_k^t \cdot \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\} \cup \{\tilde{h}_u^{k-1,t} : u \in \mathcal{O}_v^t\}).$$

Note that $\tilde{h}_u^{k-1,t} = h_u^{k-1,\tau_u}$, and the estimation error $\left\| \underline{h_v^{k,t}} - h_v^{k,t} \right\|$ depends on the "staleness" of the historical embedding $h_u^{k-1,\tau_u}$ compared to the real embedding $h_u^{k-1,t}$, where $u \in \mathcal{O}_v^t$ is an out-batch node and $\tau_u$ denotes the last iteration that node $u \in \mathcal{O}_v^t$ is sampled, i.e. $u \in \mathcal{D}_{\tau_u}$. Proposition 1 provides an upper bound of the approximation error of one node $v$ by using GNNAutoScale under some conditions.

**Proposition 1.** *Assume that the activation function $\sigma(\cdot)$ is $C_\sigma$-Lipschitz continuous and the aggregator $\mathcal{A}(\cdot)$ is $C_\mathcal{A}$-Lipschitz continuous, the weight $W_k^t$ is bounded as $\|W_k^t\| \leq C_k$. Then, the estimation error $\left\| h_v^{k,t} - \underline{h_v^{k,t}} \right\|$ of GNNAu-*

*toScale is upper bounded by its "staleness". Formally,*

$$\left\| h_v^{k,t} - \underline{h_v^{k,t}} \right\| \leq C_\sigma C_k C_\mathcal{A} \underbrace{\sum_{u \in \mathcal{O}_v^{k,t}} \left\| h_u^{k-1,\tau_u} - h_u^{k-1,t} \right\|}_{\clubsuit},$$

*where $\tau_u$ denotes the last iteration that node $u \in \mathcal{O}_v^{k,t}$ is sampled, i.e. $u \in \mathcal{D}_{\tau_u}$.*

*Proof.* Based on the definition, the estimation error of $h_v^{k,t}$ compared to $\underline{h_v^{2,t}}$ can be upper bounded as

$$\left\| h_v^{k,t} - \underline{h_v^{k,t}} \right\| \leq C_\sigma \left\| W_k^t \cdot \left( \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\} \cup \{h_u^{k-1,\tau_u} : u \in \mathcal{O}_v^t\}) - \mathcal{A}(\{h_u^{k-1,t} : u \in \overline{\mathcal{N}}_v\}) \right) \right\|$$

$$\leq C_\sigma C_k \left\| \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\} \cup \{h_u^{k-1,\tau_u} : u \in \mathcal{O}_v^t\}) - \mathcal{A}(\{h_u^{k-1,t} : u \in \overline{\mathcal{N}}_v\}) \right\| \leq C_\sigma C_k C_\mathcal{A} \sum_{u \in \mathcal{O}_v^t} \left\| h_u^{k-1,\tau_u} - h_u^{k-1,t} \right\|.$$

$\square$

The proposition below explains why GraphFM-OB could be helpful to reduce the estimation error in some cases.

**Proposition 2.** *Assume that the activation function $\sigma(\cdot)$ is $C_\sigma$-Lipschitz continuous and the aggregator $\mathcal{A}(\cdot)$ is $C_\mathcal{A}$-Lipschitz continuous, the weight $W_k^t$ is bounded as $\|W_k^t\| \leq C_k$. Then, the estimation error $\left\| h_v^{k,t} - \underline{h_v^{k,t}} \right\|$ of GraphFM-OB is upper bounded by*

$$\left\| \tilde{h}_u^{k-1,t} - h_u^{k-1,t} \right\|$$

$$\leq C_\sigma C_k C_\mathcal{A} \underbrace{\sum_{u \in \mathcal{O}_v^t} \left( (1 - \beta_{0,k-1})^{|\mathcal{T}_u^t|} \left\| h_u^{k-1,\tau_u} - h_u^{k-1}, t \right\| + \sum_{\iota=1}^{|\mathcal{T}_u^t|} (1 - \beta_{0,k-1})^{\iota-1} \beta_{0,k-1} \left\| h_u^{k-1,t_u^\iota} - h_u^{k-1,t} \right\| \right)}_{\clubsuit}$$

$$+ C_\sigma^2 C_k C_{k-1} C_\mathcal{A} \underbrace{\sum_{u \in \mathcal{O}_v^t} \left( \sum_{\iota=1}^{|\mathcal{T}_u^t|} (1 - \beta_{0,k-1})^{\iota-1} \beta_{0,k-1} \left\| \mathcal{A}(\{h_w^{k-2,t_u^\iota} : w \in \mathcal{S}_u^{t_u^\iota}\}) - \mathcal{A}(\{h_w^{k-2,t_u^\iota} : w \in \overline{\mathcal{N}}_u\}) \right\| \right)}_{\heartsuit}$$

$$+ \beta_{0,k-1} C_\sigma^2 C_k C_{k-1} C_\mathcal{A} \underbrace{\sum_{u \in \mathcal{O}_v^t} \left( \left\| \mathcal{A}(\{h_w^{k-2,t} : w \in \mathcal{S}_u^t\}) - \mathcal{A}(\{h_w^{k-2,t} : w \in \overline{\mathcal{N}}_u\}) \right\| \right)}_{\spadesuit},$$

*where $\tau_u$ ($\tau_u < t$) is the last iteration before $t$ that $u$ is sampled, i.e. $u \in \mathcal{D}_{\tau_u}$. Besides, for each $u \in \mathcal{O}_v^t$, there exists a sub-sequence $\mathcal{T}_u^t$ of $\{\tau_u, \tau_u + 1, \ldots, t\}$ satisfying $\mathcal{S}_u^{t_u^\iota} \neq \emptyset$ for every $\iota = 1, \ldots, |\mathcal{T}_u^t|$, where $t_u^\iota$ is the $\iota$-th element in $\mathcal{T}_u^t$.*

**Remark 1.** *By comparing Proposition 1 of GNNAutoScale and Proposition 2 of GraphFM-OB, it can be seen that the $\clubsuit$ term of GraphFM-OB improves upon the $\heartsuit$ term of GNNAutoScale because $h_u^{k-1,t_u^\iota}$ is less out-dated w.r.t. $h_u^{k-1,t}$ compared to $h_u^{k-1,\tau_u}$. As a trade-off, GraphFM-OB also introduces the two extra terms $\heartsuit$ and $\spadesuit$, which could be controlled if $\mathcal{S}_u$ and $\overline{\mathcal{N}}_u$ has more common elements. To ensure this, one might replace the condition in line 8 of Algorithm 2 by $(v \notin \mathcal{D}_t) \wedge (|\mathcal{D}_t \cap \overline{\mathcal{N}}_v| > c)$, where $c > 0$ is a threshold.*

*Proof.* According to the update rule of GraphFM-OB, we can derive that

$$\left\| h_v^{k,t} - \underline{h_v^{k,t}} \right\|$$

$$= \left\| \sigma \left( W_k^t \cdot \left( \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\} \cup \{\tilde{h}_u^{k-1,t} : u \in \mathcal{O}_v^t\}) \right) \right) - \sigma(W_k^t \cdot \mathcal{A}(\{h_u^{k-1,t} : u \in \overline{\mathcal{N}}_v\})) \right\|$$

$$\leq C_\sigma C_k \left\| \mathcal{A}(\{h_u^{k-1,t} : u \in \mathcal{S}_v^t\} \cup \{\tilde{h}_u^{k-1,t} : u \in \mathcal{O}_v^t\}) - \mathcal{A}(\{h_u^{k-1,t} : u \in \overline{\mathcal{N}}_v\}) \right\|$$

$$\leq C_\sigma C_k C_\mathcal{A} \sum_{u \in \mathcal{O}_v^t} \left\| \tilde{h}_u^{k-1,t} - h_u^{k-1,t} \right\|.$$

We denote that $\tau_u$ ($\tau_u < t$) is the last iteration before $t$ that $u$ is sampled, i.e. $u \in \mathcal{D}_{\tau_u}$. Besides, for each $u \in \mathcal{O}_v^t$, there exists a sub-sequence $\mathcal{T}_u^t$ of $\{\tau_u, \tau_u + 1, \ldots, t\}$ satisfying $\mathcal{S}_u^{t_u^\iota} \neq \emptyset$ for every $\iota = 1, \ldots, |\mathcal{T}_u^t|$, where $t_u^\iota$ is the $\iota$-th element in $\mathcal{T}_u^t$. Thus, the update rule of historical embeddings in GraphFM-OB leads to

$$
\left\| \tilde{h}_u^{k-1,t} - h_u^{k-1,t} \right\|
$$
$$
= \left\| (1 - \beta_{0,k-1}) \tilde{h}_u^{k-1,t_u^{|\mathcal{T}_u^t|-1}} + \beta_{0,k} \sigma(W_{k-1}^t \cdot \mathcal{A}(\{h_w^{k-2,t} : w \in \mathcal{S}_u^t\})) - \sigma(W_{k-1}^t \cdot \mathcal{A}(\{h_w^{k-2,t} : w \in \overline{\mathcal{N}}_u\})) \right\|
$$
$$
\leq (1 - \beta_{0,k-1})^{|\mathcal{T}_u^t|} \left\| h_u^{k-1,\tau_u} - h_u^{k-1}, t \right\| + \sum_{\iota=1}^{|\mathcal{T}_u^t|} (1 - \beta_{0,k-1})^{\iota-1} \beta_{0,k-1} \left\| h_u^{k-1,t_u^\iota} - h_u^{k-1,t} \right\|
$$
$$
+ C_\sigma C_{k-1} \sum_{\iota=1}^{|\mathcal{T}_u^t|} (1 - \beta_{0,k-1})^{\iota-1} \beta_{0,k-1} \left\| \mathcal{A}(\{h_w^{k-2,t_u^\iota} : w \in \mathcal{S}_u^{t_u^\iota}\}) - \mathcal{A}(\{h_w^{k-2,t_u^\iota} : w \in \overline{\mathcal{N}}_u\}) \right\|
$$
$$
+ \beta_{0,k-1} C_\sigma C_{k-1} \left\| \mathcal{A}(\{h_w^{k-2,t} : w \in \mathcal{S}_u^t\}) - \mathcal{A}(\{h_w^{k-2,t} : w \in \overline{\mathcal{N}}_u\}) \right\|
$$

$\square$

# C. Convergence Analysis of GraphFM-IB

### C.1. Setting, Assumptions, and Lemmas

The task of training GNNs can be abstracted as following multi-level stochastic compositional optimization problem.

$$
\min_{\mathbf{w} \in \mathbb{R}^D} F(\mathbf{w}), \quad F(\mathbf{w}) = f_{K+1} \circ f_K \circ \ldots f_k \ldots \circ f_1(\mathbf{w}),
$$

where $f_1 : \mathbb{R}^D \mapsto \mathbb{R}^{nd_1}$, $f_k : \mathbb{R}^{nd_{k-1}} \mapsto \mathbb{R}^{nd_k}$, $k = 2, \ldots, K$, $f_{K+1} : \mathbb{R}^{nd_K} \mapsto \mathbb{R}$. Besides, each $f_k(\cdot) \in \mathbb{R}^{nd_k}$, $k = 1, \ldots, K$ can be splitted into $n$ blocks of $d_k$ consecutive coordinates. We denote the $i$-th block of $f_k(\cdot)$ as $f_{k,i}(\cdot) \in \mathbb{R}^{d_k}$. We make the following assumption.

**Assumption 1.** *Assume that $f_{k,i}$ is $L_f$-Lipschitz continuous while $\nabla f_{k,i}$ is $L_g$-Lipschitz continuous for $k = 1, \ldots, K$. Besides, $f_{K+1}$ is $L_f$-Lipschitz continuous while $\nabla f_{K+1}$ is $L_g$-Lipschitz continuous.*

**Lemma 1.** *(Balasubramanian et al., 2020) Given Assumption 1, $F$ is $L_F$-smooth, where $L_F := L_f^{2K+1} L_g \sum_{k=1}^{K+1} \frac{1}{L_f^k}$.*

The full gradient of $F(\mathbf{w})$ can be computed as

$$
\nabla F(\mathbf{w}) = \nabla f_1(\mathbf{w}) \ldots \nabla f_K(\mathbf{y}_{K-1}) \nabla f_{K+1}(\mathbf{y}_K), \quad \mathbf{y}_k = f_k \circ \ldots \circ f_1(\mathbf{w}).
$$

At each of the $k$-th layer ($k = 1, \ldots, K$), we sample a batch of nodes $\mathcal{B}_k \subseteq \{1, \ldots, n\}$. For each node $i \in \mathcal{B}_k$, we approximate the function value $f_{k,i}(\cdot)$ and the Jacobian $\nabla f_{k,i}(\cdot)$ by using stochastic estimators $\hat{f}_{k,i}(\cdot)$ and $\hat{\nabla} f_{k,i}(\cdot)$ by sampling the neighborhood of node $i$. At the $(K+1)$-th layer, we approximate $\nabla f_{K+1}(\cdot)$ by $\hat{\nabla} f_{K+1}(\cdot)$.

**Assumption 2.** *We assume that $\mathbb{E}\left[\hat{f}_{k,i}(\cdot)\right] = f_{k,i}(\cdot)$, $\mathbb{E}\left[\left\| \hat{f}_{k,i}(\cdot) - f_{k,i}(\cdot) \right\|^2\right] \leq \sigma_f^2$ and $\mathbb{E}\left[\hat{\nabla} f_{k,i}(\cdot)\right] = \nabla f_{k,i}(\cdot)$, $\left\| \hat{\nabla} f_{k,i}(\cdot) \right\|^2 \leq \sigma_g^2$ for $1 \leq i \leq K$. Besides, $\mathbb{E}\left[\hat{\nabla} f_{K+1}(\cdot)\right] = \nabla f_{K+1}(\cdot)$, $\left\| \hat{\nabla} f_{K+1}(\cdot) \right\|^2 \leq \sigma_g^2$.*

In each iteration, we update the model parameter $\mathbf{w}$ using the following rule.

$$
\mathbf{u}_{k,i}^t = \begin{cases} (1 - \beta_{0,k})\mathbf{u}_{k,i}^{t-1} + \beta_{0,k}\hat{f}_{k,i}(\mathbf{u}_{k-1,i}^{t-1}), & i \in \mathcal{B}_k^t \\ \mathbf{u}_{k,i}^{t-1}, & i \notin \mathcal{B}_k^t \end{cases}, \quad \mathbf{u}_k^t = \begin{bmatrix} \mathbf{u}_{k,1}^t \\ \vdots \\ \mathbf{u}_{k,n}^t \end{bmatrix}, \quad k = 1, \ldots, K, \quad \mathbf{u}_0^t = \mathbf{w}^t, \quad (15)
$$

$$
\mathbf{m}^t = (1 - \beta_1)\mathbf{m}^{t-1} + \beta_1 \left( \prod_{k=1}^K \hat{g}_k^t \right) \hat{\nabla} f_{K+1}(\mathbf{u}_K^t), \quad \hat{g}_k^t = \sum_{i=1}^n \frac{\mathbb{I}[i \in \mathcal{B}_k^t]n}{B} \hat{\nabla} f_{k,i}(\mathbf{u}_{k-1}^t)\mathbf{I}_{k,i}, \quad (16)
$$

$$
\mathbf{v}^t = (1 - \beta_2)\mathbf{v}^{t-1} + \beta_2 \left( \left( \prod_{k=1}^K \hat{g}_k^t \right) \hat{\nabla} f_{K+1}(\mathbf{u}_K^t) \right)^2, \quad (17)
$$

$$
\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\mathbf{m}^t}{\sqrt{\mathbf{v}^t} + \epsilon_0}, \quad (18)
$$

where $\mathbf{I}_{k,i} \in \mathbb{R}^{d_k \times nd_k}$ is the indicator matrix that only has one non-zero block (i.e., the $i$-th block) and $B = |\mathcal{B}_k^t|$. Note that only the sampled blocks in $\hat{g}_k^t$ could be non-zero while the other blocks are padded with zeros.

**Remark 2.** *When specialized to the GNN training task, the described update rule is equivalent to GraphFM-IB (Algorithm 1). The scaling factors $\frac{n^K}{B^K}$ in $\mathbf{m}^t$ and $\mathbf{v}^t$ cancel each other out if we re-define $\mathbf{m}^0, \sqrt{\mathbf{v}^0}, \epsilon_0$ to be $\frac{n^K}{B^K}$ times larger.*

**Assumption 3.** *There exist $c_1, c_u > 0$ such that $c_l \leq \left\| \frac{1}{\sqrt{\mathbf{v}_t} + \epsilon_0} \right\| \leq c_u$.*

**Lemma 2** (Lemma 5 in Guo et al. 2021)**.** *For $\eta \leq \frac{c_l}{2c_u^2 L_F}$, we have:*

$$
F(\mathbf{w}^{t+1}) \leq F(\mathbf{w}^t) + \frac{\eta c_u}{2} \left\| \nabla F(\mathbf{w}^t) - \mathbf{m}^t \right\|^2 - \frac{\eta c_l}{2} \left\| \nabla F(\mathbf{w}^t) \right\|^2 - \frac{\eta c_l}{4} \left\| \mathbf{m}^t \right\|^2.
$$

We use $\mathcal{F}_t$ to denote all randomness occurred up to (include) the $t$-th iteration of any algorithm. We define $\Phi^t :=$ $\left\| \nabla F(\mathbf{w}^t) - \mathbf{m}^t \right\|^2$, $\Upsilon_k^t := \left\| f_k(\mathbf{u}_t^{k-1}) - \mathbf{u}_k^t \right\|^2$, $\Delta^t := \left\| \prod_{k=1}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^t) - \left( \prod_{k=1}^K \hat{g}_k^t \right) \hat{f}_{K+1}(\mathbf{u}_K^t) \right\|^2$.

**Lemma 3.** *For $\mathbf{m}^t$ following (16), we have*

$$
\mathbb{E}\left[ \Phi^{t+1} \mid \mathcal{F}_t \right] \leq (1 - \beta_1)\Phi^t + \frac{4\eta^2 c_u^2 L_F^2}{\beta_1} \left\| \mathbf{m}^t \right\|^2 + 4\beta_1 K \left( \sum_{k=1}^K C_k^2 \Upsilon_k^{t+1} \right) + \beta_1^2 C_\Delta, \quad (19)
$$

*where $C_k := L_f^K L_g(1 + L_f + \ldots + L_f^{K-k})$, $C_\Delta := \sum_{k=1}^K \frac{n^{2k} L_f^{2(K+1-k)} \sigma^{2k}}{B^k} + \frac{n^{2K} \sigma_g^{2(K+1)}}{B^K}$*

*Proof.* Based on the update rule of $\mathbf{m}^t$, we have

$$
E\left[ \Phi^{t+1} \mid \mathcal{F}_t \right] = \mathbb{E}\left[ \left\| \nabla F(\mathbf{w}^{t+1}) - \mathbf{m}^{t+1} \right\|^2 \mid \mathcal{F}_t \right]
$$

$$
= \mathbb{E}\left[ \left\| \nabla F(\mathbf{w}^{t+1}) - (1 - \beta_1)\mathbf{m}^t - \beta_1 \left( \prod_{k=1}^K \hat{g}_k^t \right) \hat{f}_{K+1}(\mathbf{u}_K^t) \right\|^2 \mid \mathcal{F}_t \right]
$$

$$
= \mathbb{E}\Big[ \Big\| (1 - \beta_1)(\nabla F(\mathbf{w}^t) - \mathbf{m}^t) + (1 - \beta_1)(\nabla F(\mathbf{w}^{t+1}) - \nabla F(\mathbf{w}^t))
$$

$$
+ \beta_1 \left( \prod_{k=1}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^t) - \left( \prod_{k=1}^K \hat{g}_k^t \right) \hat{f}_{K+1}(\mathbf{u}_K^t) \right) + \beta_1 \left( \nabla F(\mathbf{w}^{t+1}) - \prod_{k=1}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^t) \right) \Big\|^2 \mid \mathcal{F}_t \Big]
$$

$$
\leq (1 - \beta_1)\Phi^t + \frac{4\eta^2 c_u^2 L_F^2}{\beta_1} \left\| \mathbf{m}^t \right\|^2 + 4\beta_1 \mathbb{E}\left[ \left\| \nabla F(\mathbf{w}^{t+1}) - \prod_{k=1}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^t) \right\|^2 \mid \mathcal{F}_t \right] + \beta_1^2 \mathbb{E}\left[ \Delta^{t+1} \mid \mathcal{F}_t \right].
$$

Note that

$$\left\| \nabla F(\mathbf{w}^{t+1}) - \prod_{k=1}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^t) \right\| \le L_f^K L_g \sum_{k=1}^{K} \left\| \mathbf{y}_k^{t+1} - \mathbf{u}_k^{t+1} \right\|,$$

where $\mathbf{y}_k^{t+1} := f_k \circ f_{k-1} \ldots f_1(\mathbf{w}^{t+1})$. Besides, we also have

$$\left\| \mathbf{y}_k^{t+1} - \mathbf{u}_k^{t+1} \right\| \le \sum_{j=1}^{k} L_f^{k-j} \left\| f_j(\mathbf{u}_{j-1}^{t+1}) - \mathbf{u}_j^{t+1} \right\|.$$

Then,

$$\left\| \nabla F(\mathbf{w}^{t+1}) - \prod_{k=1}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^t) \right\|^2 \le K \left( \sum_{k=1}^{K} C_k^2 \Upsilon_k^{t+1} \right),$$

where $\Upsilon_k^t := \left\| f_k(\mathbf{u}_{k-1}^t) - \mathbf{u}_k^t \right\|^2$, $C_k := L_f^K L_g(1 + L_f + \ldots + L_f^{K-k})$.

Based on the definition of $\Delta^t$, we have

$$\mathbb{E}\left[ \Delta^{t+1} \mid \mathcal{F}_t \right] = \mathbb{E}\left[ \left\| \prod_{k=1}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^{t+1}) - \hat{g}_1^{t+1} \prod_{k=2}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^{t+1}) \right\|^2 \mid \mathcal{F}_t \right]$$

$$+ \mathbb{E}\left[ \left\| \hat{g}_1^{t+1} \prod_{k=2}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^{t+1}) - \hat{g}_1^{t+1} \hat{g}_2^{t+1} \prod_{k=3}^{K+1} \nabla f_k(\mathbf{u}_{k-1}^{t+1}) \right\|^2 \mid \mathcal{F}_t \right]$$

$$\ldots$$

$$+ \mathbb{E}\left[ \left\| \left( \prod_{k=1}^{K} \hat{g}_k^{t+1} \right) \nabla f_{K+1}(\mathbf{u}_K^{t+1}) - \left( \prod_{k=1}^{K} \hat{g}_k^{t+1} \right) \hat{\nabla} f_{K+1}(\mathbf{u}_K^{t+1}) \right\|^2 \mid \mathcal{F}_t \right]$$

$$\le \frac{n^2 \sigma_g^2 L_f^{2K}}{B} + \frac{n^4 \sigma_g^4 L_f^{2(K-1)} L_f^{2(K-1)}}{B^2} + \ldots + \frac{n^{2K} \sigma_g^{2K} L_f^2}{B^K} + \frac{n^{2K} \sigma_g^{2(K+1)}}{B^K}$$

$$= \underbrace{\sum_{k=1}^{K} \frac{n^{2k} L_f^{2(K+1-k)} \sigma^{2k}}{B^k} + \frac{n^{2K} \sigma_g^{2(K+1)}}{B^K}}_{:=C_\Delta}.$$

$\square$

**Lemma 4.** *For $0 < \beta_{0,k} \le 1$ and $\Upsilon_k^t := \left\| \mathbf{u}_k^t - f_k(\mathbf{u}_{k-1}^t) \right\|^2$, we have*

$$\mathbb{E}\left[ \Upsilon_k^{t+1} \right] \le \left( 1 - \frac{\beta_{0,k} B}{2n} \right) \mathbb{E}\left[ \Upsilon_k^t \right] + \beta_{0,k}^2 n \sigma_f^2 + \begin{cases} \frac{5\eta^2 c_u^2 n^2 L_f^2 \|\mathbf{m}^t\|^2}{B\beta_{0,k}}, & k = 1 \\ \frac{5\beta_{0,k-1}^2 n^2 \sigma_f^2 L_f^2}{\beta_{0,k}} + \frac{5\beta_{0,k-1}^2 n L_f^2}{\beta_{0,k}} \mathbb{E}\left[ \Upsilon_{k-1}^t \right], & 2 \le k \le K. \end{cases} \tag{20}$$

*Proof.* According to (15), we can derive that

$$\mathbb{E}\left[ \Upsilon_k^{t+1} \right] = \mathbb{E}\left[ \left\| \mathbf{u}_k^{t+1} - f_k(\mathbf{u}_{k-1}^{t+1}) \right\|^2 \right]$$

$$= \mathbb{E}\left[ \sum_{k=1}^{n} \left\| \mathbf{u}_{k,i}^{t+1} - f_{k,i}(\mathbf{u}_{k-1}^{t+1}) \right\|^2 \right]$$

$$= \mathbb{E}\left[ \sum_{k=1}^{n} \frac{B}{n} \underbrace{\left\| (1 - \beta_{0,k}) \mathbf{u}_{k,i}^t + \beta_{0,k} \hat{f}_{k,i}(\mathbf{u}_{k-1}^t) - f_{k,i}(\mathbf{u}_{k-1}^{t+1}) \right\|^2}_{\text{⑤}} + \sum_{k=1}^{n} (1 - \frac{B}{n}) \underbrace{\left\| \mathbf{u}_{k,i}^t - f_{k,i}(\mathbf{u}_{k-1}^{t+1}) \right\|^2}_{\text{⑥}} \right]$$

The first term on the R.H.S. can be bounded as

$$\mathbb{E}\left[\|⑤\|^2\right] = \mathbb{E}\left[\left\|(1-\beta_{0,k})(\mathbf{u}_{k,i}^t - f_{k,i}(\mathbf{u}_{k-1}^t)) + (f_{k,i}(\mathbf{u}_{k-1}^t) - f_{k,i}(\mathbf{u}_{k-1}^{t+1})) + \beta_{0,k}(\hat{f}_{k,i}(\mathbf{u}_{k-1}^t) - f_{k,i}(\mathbf{u}_{k-1}^t))\right\|^2\right]$$

$$\leq (1-\beta_{0,k})\mathbb{E}\left[\left\|\mathbf{u}_{k,i}^t - f_{k,i}(\mathbf{u}_{k-1}^t)\right\|^2\right] + \frac{2L_f^2}{\beta_{0,k}}\mathbb{E}\left[\left\|\mathbf{u}_{k-1}^{t+1} - \mathbf{u}_{k-1}^t\right\|^2\right] + \beta_{0,k}^2\sigma_f^2.$$

If $\beta \leq \frac{n}{B}$, we have

$$\mathbb{E}\left[\|⑥\|^2\right] \leq \left(1 + \frac{\beta_{0,k}B}{2n}\right)\mathbb{E}\left[\left\|\mathbf{u}_{k,i}^t - f_{k,i}(\mathbf{u}_{k-1}^t)\right\|^2\right] + \frac{3nL_f^2}{\beta_{0,k}B}\mathbb{E}\left[\left\|\mathbf{u}_{k-1}^{t+1} - \mathbf{u}_{k-1}^t\right\|^2\right].$$

Then,

$$\mathbb{E}\left[\Upsilon_k^{t+1}\right] \leq \left(1 - \frac{\beta_{0,k}B}{2n}\right)\mathbb{E}\left[\Upsilon_k^t\right] + \frac{5n^2L_f^2}{B\beta_{0,k}}\mathbb{E}\left[\left\|\mathbf{u}_{k-1}^{t+1} - \mathbf{u}_{k-1}^t\right\|^2\right] + \beta_{0,k}^2n\sigma_f^2.$$

When $k = 1$, we have $\mathbb{E}\left[\left\|\mathbf{u}_{k-1}^{t+1} - \mathbf{u}_{k-1}^t\right\|^2 \mid \mathcal{F}_t\right] = \left\|\mathbf{w}^{t+1} - \mathbf{w}^t\right\|^2 \leq \eta^2 c_u^2 \left\|\mathbf{m}^t\right\|^2$. When $2 \leq k \leq K$, consider that $\mathbf{u}_{k-1,i}^{t+1} = \mathbf{u}_{k-1,i}^t$ if $i \notin \mathcal{B}_{k-1}^{t+1}$.

$$\mathbb{E}\left[\left\|\mathbf{u}_{k-1}^{t+1} - \mathbf{u}_{k-1}^t\right\|^2\right]$$

$$= \mathbb{E}\left[\sum_{i\in\mathcal{B}_{k-1}^{t+1}}\left\|\mathbf{u}_{k-1,i}^{t+1} - \mathbf{u}_{k-1,i}^t\right\|^2\right] = \beta_{0,k-1}^2\mathbb{E}\left[\sum_{i\in\mathcal{B}_{k-1}^{t+1}}\left\|\hat{f}_{k-1,i}(\mathbf{u}_{k-2}^t) - \mathbf{u}_{k-1,i}^t\right\|^2\right]$$

$$= \beta_{0,k-1}^2\mathbb{E}\left[\sum_{i\in\mathcal{B}_{k-1}^{t+1}}\left\|\hat{f}_{k-1,i}(\mathbf{u}_{k-2}^t) - f_{k-1,i}(\mathbf{u}_{k-2}^t)\right\|^2\right] + \beta_{0,k-1}^2\mathbb{E}\left[\sum_{i\in\mathcal{B}_{k-1}^{t+1}}\left\|f_{k-1,i}(\mathbf{u}_{k-2}^t) - \mathbf{u}_{k-1,i}^t\right\|^2\right]$$

$$\leq \beta_{0,k-1}^2B\sigma_f^2 + \frac{\beta_{0,k-1}^2B}{n}\mathbb{E}\left[\left\|f_{k-1}(\mathbf{u}_{k-2}^t) - \mathbf{u}_{k-1}^t\right\|^2\right] = \beta_{0,k-1}^2B\sigma_f^2 + \frac{\beta_{0,k-1}^2B}{n}\mathbb{E}\left[\Upsilon_{k-1}^t\right].$$

Thus, we have

$$\mathbb{E}\left[\Upsilon_k^{t+1}\right] \leq \left(1 - \frac{\beta_{0,k}B}{2n}\right)\mathbb{E}\left[\Upsilon_k^t\right] + \beta_{0,k}^2n\sigma_f^2 + \begin{cases} \frac{5\eta^2c_u^2n^2L_f^2\left\|\mathbf{m}^t\right\|^2}{B\beta_{0,k}}, & k = 1 \\ \frac{5\beta_{0,k-1}^2n^2\sigma_f^2L_f^2}{\beta_{0,k}} + \frac{5\beta_{0,k-1}^2nL_f^2}{\beta_{0,k}}\mathbb{E}\left[\Upsilon_{k-1}^t\right], & 2 \leq k \leq K. \end{cases}$$

$\square$

## C.2. Proof of Theorem 1

*Proof.* Take expectation on both sides of Lemma 2 and telescope the recursion from iteration 1 to $T$.

$$\sum_{t=1}^T\mathbb{E}\left[\left\|\nabla F(\mathbf{w}^t)\right\|^2\right] \leq \frac{2(F(\mathbf{w}^t) - F^*)}{\eta c_l} + \frac{c_u}{c_l}\sum_{t=1}^T\mathbb{E}\left[\Phi^t\right] - \frac{1}{2}\sum_{t=1}^T\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right], \tag{21}$$

where $F^*$ is the global lower bound of $F(\mathbf{w})$. Using the tower property of conditional expectation and telescoping (19) leads to

$$\sum_{t=1}^T\mathbb{E}\left[\Phi^t\right] \leq \frac{\mathbb{E}\left[\Phi^1\right]}{\beta_1} + \frac{4\eta^2c_u^2L_F^2}{\beta_1^2}\sum_{t=1}^T\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right] + 4K\sum_{t=1}^T\sum_{k=1}^K C_k^2\Upsilon_k^{t+1} + \beta_1TC_\Delta. \tag{22}$$

Telescope (20) from iteration 1 to $T$.

$$\sum_{t=1}^{T} \mathbb{E}\left[\Upsilon_k^t\right] \leq \frac{2n\mathbb{E}\left[\Upsilon_k^1\right]}{B\beta_{0,k}} + \frac{2Tn^2\sigma_f^2\beta_{0,k}}{B} + \frac{10Tn^3\sigma_f^2L_f^2\beta_{0,k-1}^2}{B\beta_{0,k}^2} + \frac{10n^2L_f^2\beta_{0,k-1}^2}{B\beta_{0,k}^2}\sum_{t=1}^{T}\mathbb{E}\left[\Upsilon_{k-1}^t\right], \quad 2 \leq k \leq K,$$

$$\sum_{t=1}^{T} \mathbb{E}\left[\Upsilon_k^t\right] \leq \frac{2n\mathbb{E}\left[\Upsilon_k^1\right]}{B\beta_{0,k}} + \frac{2Tn^2\sigma_f^2\beta_{0,k}}{B} + \frac{10\eta^2 c_u^2 n^3 L_f^2}{B^2\beta_{0,k}^2}\sum_{t=1}^{T}\left\|\mathbf{m}^t\right\|^2, \quad k = 1.$$

We can further derive that

$$\sum_{t=1}^{T} \mathbb{E}\left[\Upsilon_k^t\right] \leq \sum_{\iota=1}^{k} \frac{2n\beta_{0,\iota}\mathbb{E}\left[\Upsilon_\iota^1\right]}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \sum_{\iota=1}^{k} \frac{2Tn^2\sigma_f^2\beta_{0,\iota}^3}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota}$$

$$+ \sum_{\iota=2}^{k} \frac{10Tn^3\sigma_f^2L_f^2\beta_{0,\iota-1}^2}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \left(\frac{10n^2L_f^2}{B}\right)^{k-1}\frac{10\eta^2 c_u^2 n^3 L_f^2}{B^2\beta_{0,k}^2}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right].$$

Then,

$$\sum_{t=1}^{T}\sum_{i=1}^{K} C_k^2 \Upsilon_k^{t+1}$$

$$\leq \sum_{k=1}^{K}\sum_{\iota=1}^{k} \frac{2C_k^2 n\beta_{0,\iota}\mathbb{E}\left[\Upsilon_\iota^2\right]}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \sum_{k=1}^{K}\sum_{\iota=1}^{k} \frac{2C_k^2 Tn^2\sigma_f^2\beta_{0,\iota}^3}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota}$$

$$+ \sum_{i=1}^{K}\sum_{\iota=2}^{k} \frac{10C_k^2 Tn^3\sigma_f^2L_f^2\beta_{0,\iota-1}^2}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \left(\sum_{t=2}^{T+1}\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right]\right)\sum_{k=1}^{K}C_k^2\left(\frac{10n^2L_f^2}{B}\right)^{k-1}\frac{10\eta^2 c_u^2 n^3 L_f^2}{B^2\beta_{0,k}^2}.$$

Plug the inequality above into (22).

$$\sum_{t=1}^{T}\mathbb{E}\left[\Phi^t\right] \leq \frac{\mathbb{E}\left[\Phi^1\right]}{\beta_1} + \frac{4\eta^2 c_u^2 L_F^2}{\beta_1^2}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right] + \left(\sum_{t=2}^{T+1}\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right]\right)\sum_{k=1}^{K}C_k^2\left(\frac{40Kn^2L_f^2}{B}\right)^{k-1}\frac{10\eta^2 c_u^2 n^3 L_f^2}{B^2\beta_{0,k}^2}$$

$$+ \sum_{k=1}^{K}\sum_{\iota=1}^{k} \frac{8KC_k^2 n\beta_{0,\iota}\mathbb{E}\left[\Upsilon_\iota^2\right]}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \sum_{k=1}^{K}\sum_{\iota=1}^{k} \frac{8KC_k^2 Tn^2\sigma_f^2\beta_{0,\iota}^3}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota}$$

$$+ \sum_{k=1}^{K}\sum_{\iota=2}^{k} \frac{40KC_k^2 Tn^3\sigma_f^2L_f^2\beta_{0,\iota-1}^2}{B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \beta_1 T C_\Delta.$$

Next, we plug the inequality above into (21) and divide $T$ on both sides.

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\nabla F(\mathbf{w}^t)\right\|^2\right]$$

$$\leq \frac{2(F(\mathbf{w}^t) - F^*)}{\eta T c_l} + \frac{c_u\mathbb{E}\left[\Phi^1\right]}{\beta_1 T c_l} + \sum_{k=1}^{K}\sum_{\iota=1}^{k} \frac{8KC_k^2 nc_u\beta_{0,\iota}\mathbb{E}\left[\Upsilon_\iota^2\right]}{c_l B\beta_{0,k}^2 T}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota}$$

$$- \frac{1}{2T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right] + \frac{4\eta^2 c_u^3 L_F^2}{c_l\beta_1^2 T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right] + \left(\frac{1}{T}\sum_{t=2}^{T+1}\mathbb{E}\left[\left\|\mathbf{m}^t\right\|^2\right]\right)\sum_{k=1}^{K}C_k^2\left(\frac{40Kn^2L_f^2}{B}\right)^{k-1}\frac{10\eta^2 c_u^3 n^3 L_f^2}{c_l B^2\beta_{0,k}^2}$$

$$+ \sum_{k=1}^{K}\sum_{\iota=1}^{k} \frac{8Kc_u C_k^2 n^2\sigma_f^2\beta_{0,\iota}^3}{c_l B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \sum_{k=1}^{K}\sum_{\iota=2}^{k} \frac{40c_u KC_k^2 n^3\sigma_f^2L_f^2\beta_{0,\iota-1}^2}{c_l B\beta_{0,k}^2}\left(\frac{10n^2L_f^2}{B}\right)^{k-\iota} + \frac{\beta_1 c_u C_\Delta}{c_l}.$$

We can make $\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\left\|\nabla F(\mathbf{w}^t)\right\|^2\right] \leq \epsilon^2$ if we set $\eta = O(\epsilon^K)$, $\beta_1 = O(\epsilon^K)$, $\beta_2 \in (0,1)$, $\beta_{0,k} = O(\epsilon^{K-k})$, $1 \leq k \leq K$, and $T = O(\epsilon^{-(K+2)})$. $\qquad\square$