`BLB-HGNN`: Bag of Little Bootstraps for Training Heterogeneous GNNs

Aditya T. Vadlamani*, Sama Salarian*, Saket Gurukar†^ and Srinivasan Parthasarathy*
*Department of Computer Science & Engineering, The Ohio State University, Columbus, Ohio, USA
†Microsoft, Mountain View, CA, USA
{vadlamani.12, salarian.1}@osu.edu, saket.gurukar@gmail.com, srini@cse.ohio-state.edu

***Abstract*—Graphs can model complex relational data, which makes them invaluable in numerous machine-learning applications. While the graph's structure can be efficiently represented and stored, the associated feature memory is substantial. The feature memory can be terabytes or petabytes for web-scale graphs, especially at companies like Pinterest and Google. The extra feature memory can require external storage devices with slower I/O times for data loading, making training even slower. To address the storage issue with feature memory, we aim to reduce the number of training nodes needed through sampling, reducing the storage requirement. However, the problem is that simply training a model with fewer nodes will result in worse performance. To solve that problem, we propose `BLB-HGNN`, a training algorithm based on the Bag of Little Bootstraps. `BLB-HGNN` independently trains several replicas of the architecture on different subsamples of the data. For each training epoch, our `blb-sampler` creates a bootstrap resampling of the data for the replica to train on. The trained replicas are merged using parameter averaging and then fine-tuned for inference. We conduct experiments with the OGB_MAG and MAG240M datasets to demonstrate the effectiveness of `BLB-HGNN` over simple training. We also conduct experiments on the impact of different sampling methods and model merging techniques. With almost no additional runtime cost, `BLB-HGNN` consistently provides a performance boost of up to 5% compared to standard training with the same training budget. Applying a non-uniform sampling method, such as Personalized PageRank or Spread Sampling, further improves performance. Furthermore, `BLB-HGNN` can achieve performance close to full dataset training with less than 50% of the training data on specific models. To our knowledge, this is the first work addressing the storage problem and uses Bag of Little Bootstraps for HGNN training.**

***Index Terms*—Scalability, Graph Machine Learning, Bag of Little Bootstraps**

I. Introduction

Graphs, or networks, are ubiquitous for modeling relational data across many domains. Examples include social networks [1], biological networks for modeling protein-protein interactions [2], and nearest neighbor graphs in databases [3]. To address graph-based learning tasks, Graph Neural Networks (GNNs) have demonstrated great success in various applications, such as recommender systems [4], [5], molecular structure prediction [6], [7], and network anomaly detection [8].

However, performing GNN training at large scales remains a challenging problem. One of the key difficulties is that standard GNN architectures, including GCN [9], GAT [10], HetGNN [11], HGCN [12], and HGT [13], often struggle to scale with large graph datasets. This is because they require not only the entire graph, but also all node features, to be available on a single machine, which can be highly storage-intensive.

For instance, the MAG240M heterogeneous graph dataset from OGB-LSC [14] requires over 200 GB of storage, with 175 GB dedicated to the node features of a single node type. If node features for all node types are considered, the total memory requirement for node features would double. For web-scale graphs with billions of nodes, the storage requirement can reach several terabytes or even petabytes, with node features occupying a large portion of the total storage. To give a concrete example, a version of Pinterest's graph and features requires more than 3 TB of space [15].

Such vast storage requirements are difficult to meet, both for solving critical tasks and for benchmarking within the field [14], [16], [17]. In industry settings with graphs containing over 100 billion nodes, a less storage-intensive training approach is highly desirable. Existing research on scaling GNNs to large graphs typically relies on expensive distributed training or feature-hungry sampling techniques, both of which still require full feature storage.

In this work, we take an orthogonal approach to training on large graphs by reducing both the number of training instances and the amount of feature storage required. This approach is beneficial because it allows scaling to large graphs whose features cannot be easily stored on a single device. Additionally, it applies to heterogeneous graphs since the training nodes consist of a single node type, allowing the GNN to handle complexities related to node heterogeneity. Given that real-world graphs are typically heterogeneous, we focus on this setting.

Several challenges arise with this approach, including:

1) How do we efficiently sample the nodes required for training?
2) How do we train the model on the sampled dataset without sacrificing performance?

To address these challenges, we propose `BLB-HGNN`. For the first challenge, we investigate the impact of various sampling techniques, including Personalized PageRank [18] and Spread Sampling [19]. To tackle the second challenge, we introduce the use of Bag of Little Bootstraps (BLB) [20] for

^This work was done when the author was a student at The Ohio State University.

Heterogeneous Graph Neural Networks (HGNNs). BLB is a computationally efficient variation of the bootstrap method that combines subsampling and bootstrapping techniques, making it well-suited for parallel processing. BLB-HGNN applies subsampling and resampling to the dataset to train multiple models, which are then combined to generate a final model for inference.

We empirically evaluate BLB−HGNN on two of the largest publicly available heterogeneous graph datasets: MAG240M and OGB_MAG. MAG240M consists of over 240 million nodes and 1.7 billion edges. Storing the graph alone requires approximately 25 GB, while the provided features take up an additional 187 GB. If all node types are considered, the feature memory increases to 375 GB. Further inclusion of embeddings, such as those from metapath2vec [21] or ComplEx [22], can increase the total feature memory to $244M \cdot l \cdot b$, where l is the embedding dimension and b is the number of bytes per data type. For example, if the metapath2vec features are 128-dimensional float16 vectors, the total feature memory increases to 437.5 GB.

The key advantage of BLB−HGNN is that it allows control over the fraction of the training set used. In our experiments with MAG240M, using only 10% of the training data reduced the total feature memory for training (including neighbor features) by 75%.

Key Contributions: To summarize, our contributions are as follows:

1) To our knowledge, this is the first work to address the storage problem in training GNNs, and the first to propose using Bag of Little Bootstraps and model merging for heterogeneous GNN training.
2) We demonstrate that BLB−HGNN can improve performance by up to 5% compared to simple training with fewer nodes.
3) We show that parameter averaging with fine-tuning, when applied to model merging, outperforms other well-known ensemble techniques by over 10%.
4) We analyze the effects of different sampling methods and find that non-uniform sampling techniques, such as Personalized PageRank or Spread Sampling, can improve performance by up to 2%.

II. Preliminaries

Definition II.1. A **heterogeneous graph**, $G = (V, E, T_V, T_E, X, \phi, \psi, \varphi)$, is a graph where V, E, T_V, T_E are the sets of nodes, edges, node types, and edge types, respectively. The functions $\phi : V \to T_V$ and $\psi : E \to T_E$ map nodes and edges to node and edge types, respectively. Nodes can have *features*, where the features are elements of a feature space X. The function φ maps a node to its features.

A. Graph Sampling

When working with large graphs and limited resources, it is often desirable to sample a representative set of nodes. Selecting nodes based on "importance" (e.g., using betweenness centrality) can be computationally expensive, so efficient node sampling methods are crucial. Two popular sampling methods are *Personalized PageRank* and *Spread Sampling*.

1) Personalized PageRank: Given a graph G with $|V|$ nodes, let $\mathbf{D}$ be its degree matrix and $\mathbf{A}$ be its adjacency matrix. The standard PageRank is the unique solution to the equation:

$$\mathrm{pr}(d, \vec{p}) = (1 - d) \cdot \vec{p} + d \cdot \mathrm{pr}(d, \vec{p})\mathbf{D}^{-1}\mathbf{A},$$

where d is the *damping factor* and $\vec{p} = \left[\frac{1}{|V|}, \dots, \frac{1}{|V|}\right] \in \mathbb{R}^{|V|}$ represents a uniform distribution over the nodes. $\vec{p}$ is referred to as the *personalization vector*.

Personalized PageRank generalizes this by allowing $\vec{p}$ to represent a non-uniform distribution over the nodes. It is a random walk-based method where the final sampling depends more on the initial choice of seed nodes, making it a "local" sampling technique.

2) Spread Sampling: Spread Sampling emphasizes diversity among the sampled nodes in terms of community representation [19]. The method is guided by two hyperparameters: the *infection rate* and the *removal threshold*.

Given a set of candidate nodes $\mathbf{C}$, the infection rate determines a node's affinity for being sampled. After some nodes are sampled, the set $\mathbf{C}$ is filtered based on how many of a node's neighbors have already been sampled. The filtering uses the removal threshold as a criterion.

With these two parameters, Spread Sampling provides two benefits:

- A near-uniform sampling of nodes,
- A more spread-out set of sampled nodes that aims to capture the global structure of the graph.

Spread Sampling is a "global" sampling method, designed to sample nodes from across modular communities within a graph.

B. Bag of Little Bootstraps

Bag of Little Bootstraps (BLB) is a computationally efficient variant of the Bootstrap that combines features of subsampling and resampling. BLB performs bootstrapping on smaller subsamples of the data and averages the results to get a final estimate.

Formally, let $\mathcal{D} = \{X_1, \dots, X_n\}$ be a dataset drawn from an underlying distribution P. We subsample s subsets uniformly at random, $\mathcal{S}_1, \dots, \mathcal{S}_s \subset \mathcal{D}$, where $|\mathcal{S}_i| = b < n$. Let $P_{n,b}^{(i)}$ be the empirical distribution for $\mathcal{S}_i$. The goal is to compute some metric ξ (e.g., confidence interval or standard error) on the distribution of our estimator, denoted as $Q_n(P)$.

BLB estimates $\xi(Q_n(P))$ by averaging over the subsamples:

$$\xi(Q_n(P)) \approx \frac{1}{s} \sum_{i=1}^{s} \xi(Q_n(P_{n,b}^{(i)})),$$

where each $\xi(Q_n(P_{n,b}^{(i)}))$ is computed numerically using the bootstrap. For each $\mathcal{S}_i$, we resample n points r times, giving us $\mathcal{S}_{i_j}$ for $j = 1, \dots, r$. Averaging across the resamples provides an empirical distribution $Q_{n,i}$, from which we approximate $\xi(Q_n(P^{(i)}_{n,b}))$.

This method reduces storage costs because for each $\mathcal{S}_{i_j}$, there are at most b distinct points, with $b \ll n$, allowing significant memory savings.

C. Model Merging

A key step in the Bag of Little Bootstraps methodology is to average the estimates from the different subsamples. Extending this averaging process to models results in *model merging*. Model merging integrates the parameters or predictions of multiple models into a single model or prediction.

A classical example of this paradigm is *ensemble learning*, which combines the predictions of several weaker models to produce a more robust prediction. However, ensemble learning methods (e.g., bagging [23]) typically require multiple models for inference, leading to increased memory usage. In cases where there are many classes, voting-based approaches may need a large number of weak models to guarantee a majority vote, further inflating memory costs.

A popular model merging technique in computer vision (CV) and natural language processing (NLP) is *parameter averaging*, also known as *model soups* [24]–[28]. Using parameter averaging, we can (1) reduce the memory footprint for inference, as the final model is a single instance, and (2) allow for further fine-tuning after merging.

III. Methodology

In this section, we address the two challenges outlined in the introduction: graph sampling and training on the sampled graph. First, we provide some intuition for our proposed algorithm, `BLB−HGNN`, and then elaborate on the graph sampling process.

A. Algorithm Intuition

`BLB−HGNN` is a training framework designed to make graph neural networks (GNNs) more scalable and efficient, particularly when dealing with large, complex graphs. It is inspired by a statistical method called Bag of Little Bootstraps (BLB) and adapts this concept to the world of GNNs. The primary objective is to reduce memory and compute demands without compromising model performance.

At a high level, `BLB−HGNN` changes the way we train GNNs. Instead of training one large model on the entire dataset, it trains several smaller models, or *replicas*, on different subsets of a sampled portion of the graph. Each replica is exposed to a slightly different version of the training data, thanks to bootstrapping, enabling it to learn unique patterns. After training all the replicas, their parameters are averaged to form a single, stronger model. This final model is then fine-tuned on the sampled data to enhance its predictions.

The framework grants control over how much data to use, how many replicas to train, and how each replica's training data is chosen. Smart sampling strategies, such as *Personalized PageRank* or *Spread Sampling*, help ensure that the training data is a representative subset of the entire graph, which is particularly beneficial in large-scale settings. Since sampling and training can occur independently across replicas, the whole process is highly parallelizable, allowing for easy scaling.

Figure 1 illustrates an example of `BLB−HGNN` using half the training data and three replicas.

In the next section, we delve deeper into the algorithm, explaining how data is sampled, how the replicas are trained, and how the final model is assembled.

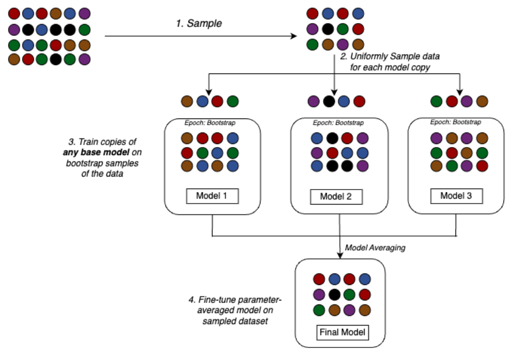
1. Sample
2. Uniformly Sample data for each model copy
3. Train copies of **any base model** on bootstrap samples of the data
Epoch: Bootstrap
Epoch: Bootstrap
Epoch: Bootstrap
Model 1
Model 2
Model 3
Model Averaging
4. Fine-tune parameter-averaged model on sampled dataset
Final Model

Fig. 1. Example of the `BLB−HGNN` training pipeline. `BLB−HGNN` is model-agnostic, meaning any underlying HGNN model can be used.

B. `BLB−HGNN` Algorithm

Algorithm 2 presents the `BLB−HGNN` algorithm. The framework takes several key parameters: $b \in (0, 1]$, the fraction of training data to use (controlling the storage required for training features); $s \in \mathbb{N}$, the number of model replicas to be trained; and $b_{\text{rep}} \in (0, 1]$, the fraction of the sampled dataset used for training each replica. The role of s and b_{rep} in `BLB−HGNN` mirrors the roles of s and b in the original BLB algorithm. In BLB, s refers to the number of subsets sampled, whereas in `BLB−HGNN`, s corresponds to the number of model replicas, each trained on a subset of the data. Both b and b_{rep} control the size of the data subsets in their respective contexts. `BLB−HGNN` proceeds as follows:

1) **Node Sampling:** The algorithm begins by sampling nodes from the training data, $\mathcal{D}_{\text{train}}$, either uniformly or with a sampling method such as PPR or Spread Sampling. For larger graphs, this stage can be performed a priori, since sampling only requires the graph structure. The resulting sampled data is denoted as $\mathcal{D}_{\text{train}_0}$, where we impose $|\mathcal{D}_{\text{train}_0}| = \lfloor b \times |\mathcal{D}_{\text{train}}| \rfloor$.
2) **Model Initialization:** Let $\mathcal{M}_1, \ldots, \mathcal{M}_s$ represent s copies of the model architecture. For each $\mathcal{M}_i$, the `train_replica` method (lines 24-29) is invoked to train the replica over r epochs. During each epoch, the `blb_sampler` method (lines 31-37) generates a bootstrap sample from $\mathcal{D}_{\text{train}_0}$.
3) **Replica Training:** The `blb_sampler` method performs the following steps: In line 33, an RNG is seeded to ensure the reproducibility of the sampling process. In line 34, a subsample of the data is drawn for $\mathcal{M}_i$, resulting in $\mathcal{D}_{\text{train}_i} = \lfloor b_{\text{rep}} \times |\mathcal{D}_{\text{train}_0}| \rfloor$. In each epoch (line

35), the RNG is reseeded to generate a new bootstrap sample, and the replica is trained on that data in line 36 using mini-batch gradient descent.

4) **Model Merging:** Once all s replicas, $\mathcal{M}_1, \dots, \mathcal{M}_s$, are trained, their parameters are averaged to create the model $\mathcal{M}_{\text{avg}} = \frac{1}{s}\sum_{i=1}^{s}\mathcal{M}_i$ (line 10).

5) **Fine-tuning:** The averaged model $\mathcal{M}_{\text{avg}}$ is then fine-tuned on the sampled data $\mathcal{D}_{\text{train}_0}$ for r_{avg} epochs, yielding the final model for inference (lines 18-22).

This process allows for efficient training and memory usage, as multiple replicas are trained on different subsets of the data, and the final model is obtained through model averaging and fine-tuning.

Fig. 2. BLB-HGNN Algorithm

```
1: **Input:** $\mathcal{D}_{\text{train}} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$, $\mathcal{D}_{\text{val}} = \{\boldsymbol{x}_i, y_i\}_{i=n+1}^{m}$, $\mathcal{D}_{\text{test}} = \{\boldsymbol{x}_i, y_i\}_{i=m+1}^{k}$, $b, b_{\text{rep}} \in (0,1]$, $s, r, r_{\text{avg}} \in \mathbb{N}$.
2: **Output:** Test Accuracy
3: models = []
4: $\mathcal{D}_{\text{train}_0} \leftarrow$ **sample**$(\mathcal{D}_{\text{train}}, b)$
5: **for** $i = 1 \dots s$ **do**
6:     $\mathcal{M}_i \leftarrow$ model_init($i$)
7:     train_replica($\mathcal{M}_i$, $\mathcal{D}_{\text{train}_0}$, $\mathcal{D}_{\text{val}}$, $i$, $r$, $b_{\text{rep}}$)
8:     models.append($\mathcal{M}_i$)
9: **end for**
10: $\mathcal{M}_{\text{avg}} \leftarrow$ avg_model_params(models)
11: train_ensemble($\mathcal{M}_{\text{avg}}$, $\mathcal{D}_{\text{train}_0}$, $\mathcal{D}_{\text{val}}$, $r_{\text{avg}}$)
12: **Return** compute_test_acc($\mathcal{M}_{\text{avg}}$, $\mathcal{D}_{\text{test}}$)
13:
14: **Function** sample ($\mathcal{D}_{\text{train}}, b$)
15: **Sample** nodes from $\mathcal{D}_{\text{train}}$ (see Section III-C)
16: **Return** $\mathcal{D}_{\text{train}_0}$
17:
18: **Function** train_ensemble ($\mathcal{M}_{\text{avg}}, \mathcal{D}_{\text{train}_0}, \mathcal{D}_{\text{val}}, r_{\text{avg}}$)
19: **for** epoch = $1 \dots r_{\text{avg}}$ **do**
20:     train($\mathcal{M}_{\text{avg}}$, $\mathcal{D}_{\text{train}_0}$)
21:     validate($\mathcal{M}_{\text{avg}}$, $\mathcal{D}_{\text{val}}$)
22: **end for**
23:
24: **Function** train_replica ($\mathcal{M}_i, \mathcal{D}_{\text{train}_0}, \mathcal{D}_{\text{val}}, i, r, b_{\text{rep}}$)
25: **for** epoch = $1 \dots r$ **do**
26:     $\mathcal{D}_{\text{blb}} \leftarrow$ blb_sampler($\mathcal{D}_{\text{train}_0}$, $i$, epoch, $b_{\text{rep}}$)
27:     train($\mathcal{M}_i$, $\mathcal{D}_{\text{blb}}$)
28:     validate($\mathcal{M}_i$, $\mathcal{D}_{\text{val}}$)
29: **end for**
30:
31: **Function** blb_sampler ($\mathcal{D}_{\text{train}_0}, i$, epoch, $b_{\text{rep}}$)
32: $n \leftarrow$ size($\mathcal{D}_{\text{train}_0}$)
33: seed($i$)
34: $\mathcal{D}_{\text{train}_i} \leftarrow$ subsample($\mathcal{D}_{\text{train}_0}$, $\lfloor n \times b_{\text{rep}} \rfloor$)
35: seed($i$ + epoch)
36: $\mathcal{D}_{\text{blb}} \leftarrow$ resample($\mathcal{D}_{\text{train}_i}$, $n$)
37: **Return** $\mathcal{D}_{\text{blb}}$
```

C. Sampling Methods

For BLB-HGNN, we explored two node sampling methods: Personalized PageRank and Spread Sampling, which approach node selection from different perspectives (local vs. global). Additionally, we examined a mixed variant—PPR-SS—which combines both local and global perspectives to investigate the impact of using both strategies together.

1) Spread Sampling: We implement the Spread Sampling algorithm as outlined in [19], treating the input graph as homogeneous. In this method, our candidate set, $\mathbf{C}$, is the set of all training nodes, and the target sample size is $\lfloor b \times |\mathcal{D}_{\text{train}}| \rfloor$, where b is the fraction of data we wish to sample. The resulting reduced training set is denoted as $\mathcal{D}_{\text{train}_0}$. For hyperparameters, we select a low infection rate (0.1) and a low removal threshold (1), which leads to more neighbor removals, thereby providing better coverage of the graph's community structure. To handle large graphs and candidate sets efficiently, we parallelized the algorithm using the Ray framework [29].

2) Personalized PageRank: In Personalized PageRank (PPR), random walks are performed with a restart probability of 0.15 from a set of seed nodes, $\mathbf{S}$. For each walk, we track the visited nodes, ensuring that we record nodes of each type. The collection of all visited nodes for a given seed node forms its neighborhood, $\mathcal{N}_{s_i}$, which includes the seed itself. The sampled training set is then defined as $\mathcal{D}_{\text{train}_0} = \bigcup_{s_i \in \mathbf{S}} \mathcal{N}_{s_i}$, which is the union of all neighborhoods of the seed nodes.

3) PPR-SS: In the PPR approach, the default is to sample seed nodes uniformly from the training data. However, in the PPR-SS variant, we enhance this by using Spread Sampling to select seed nodes that are more spread out across the graph. This mixed approach combines the strengths of both local (PPR) and global (Spread Sampling) sampling strategies.

D. Memory and Runtime Analysis

Memory: The BLB-HGNN framework reduces the number of training nodes by a factor of b, which directly impacts the memory requirements. Let the node embeddings be N-dimensional and stored using k-byte floats. By sampling a fraction b of the training nodes, the memory needed for training node features decreases by $(1-b) \cdot N \cdot k$ bytes.

Further memory savings are possible when BLB-HGNN is applied to an existing heterogeneous GNN that precomputes node neighborhoods. In such cases, neighbor sampling is not performed on the fly, reducing the number of features that need to be stored. For methods that perform neighbor sampling dynamically, the entire feature set may need to be available during training.

Figures 3 and 4 illustrate how the average feature memory required for storage decreases when BLB-HGNN is applied to the MultiBiSage model [15] on the MAG240M and OGB_MAG datasets. This is compared to the total memory required by the baseline MultiBiSage model, where $b = 1$ (i.e., no data reduction). Additionally, because BLB-HGNN trains smaller models independently and then merges them, there are further opportunities for compression on the individual replicas or the final model. To further reduce memory requirements,techniques such as quantization-aware training or structured pruning, though beyond the scope of this work, could be applied.

Runtime: BLB-HGNN is embarrassingly parallel, as the s individual model replicas can be trained independently on separate machines before the fine-tuning and inference

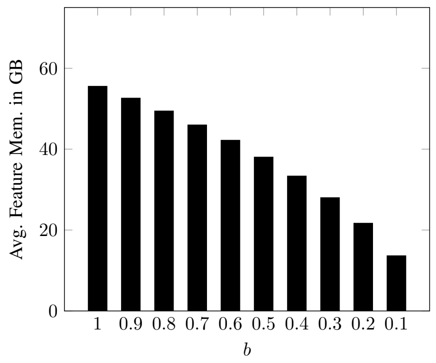

Fig. 3. Avg. Feature Memory for training MAG240M

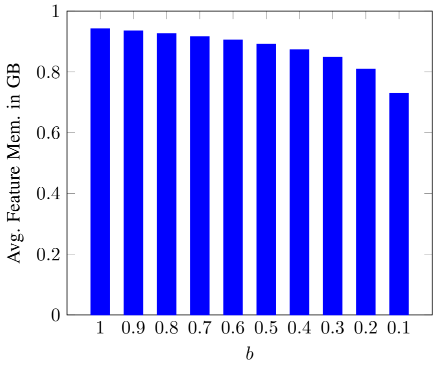

Fig. 4. Avg. Feature Memory for training OGB_MAG

phases. This parallelization can be further scaled using other techniques, such as Data and Model Parallelism.

For a fixed value of b, if we train the s model replicas independently for r epochs and fine-tune for r_{avg} epochs, then `BLB-HGNN` is $b \cdot \left(\frac{r+r_{\text{avg}}}{r_0}\right)$ times faster than simple training on the *full dataset* for r_0 epochs. In our experiments, we set $r_{\text{avg}} = r = \frac{r_0}{2}$, so for $b < 1$, we observe both runtime and memory benefits over training on the full dataset. If $r + r_{\text{avg}} = r_0$, then `BLB-HGNN` is as fast as simple training on b fraction of the data, making it an efficient alternative.

IV. Experiments

A. Setup

1) Datasets: We perform experiments on two large datasets: MAG240M [14] and OGB_MAG [16]. The MAG240M dataset is derived from the Microsoft Academic Graph (MAG) [30] and is one of the largest publicly available heterogeneous datasets. It is a heterogeneous graph with *paper*, *author*, and *institution* nodes, where the edges represent the *paper-cites-paper*, *author-writes-paper*, and *author-affiliated-with-institution* relations.

The OGB_MAG dataset shares the same node and edge types as MAG240M, with an additional *field-of-study* node type and a *paper-with-field-of-study* edge relation. Table I presents basic statistics for both datasets, including the number of labeled nodes, $|V_L|$, available for supervised learning. The task for both datasets is node classification.

For both datasets, only the *paper* node features are available. For the other node types in OGB_MAG, we used the metapath2vec features provided by PyTorch Geometric [31]. For MAG240M, we use the features from the DGL baseline [14] and the metapath2vec features used in [32].

TABLE I
Summary Statistics for Datasets

Dataset	$\|V\|$	$\|V_L\|$	$\|E\|$
MAG240M	$244,160,499$	$1,398,159$	$1,728,364,232$
OGB_MAG	$1,939,743$	$736,389$	$21,111,007$

2) Models: `BLB-HGNN` is not tied to a particular model. For our experiments, we consider two heterogeneous models: *MultiBiSage* [15] and SeHGNN [33]. *MultiBiSage* is a heterogeneous GNN developed at Pinterest and is one of the latest works to perform competitively on both industry and academic graphs, outperforming other HGNNs. It achieves this by decomposing the heterogeneous graph into bipartite graphs (graphs with a single edge type), which enables it to learn more expressive representations. SeHGNN, on the other hand, is an HGNN that performs competitively on the OGB_MAG dataset, where it ranks highly on the leaderboard.

For our main experiments and ablation studies, we used *MultiBiSage* as our primary model, while SeHGNN was used to demonstrate the generalizability of our approach.

3) Technical Details: MultiBiSage is proprietary to Pinterest, with no official public implementation available. As part of this work, we independently implemented *MultiBiSage* from scratch by following the original paper, using PyTorch Lightning [34]. [1] Additionally, we developed an optimized random walker using the Ray framework to compute node neighborhoods. For each bipartite graph, random walks were performed with a restart probability of $p = 0.15$ for up to 1000 restarts, and the 50 most common neighbors for each node type were collected.

The model hyperparameters evaluated for each dataset include embedding dimension $\{128, 256, 512\}$, number of heads $\{2, 4, 8\}$, and number of layers $\{2, 4\}$. The final model parameters for OGB_MAG were: embedding dimension = 256, transformer heads = 8, transformer layers = 2, and dropout = 0.60. For MAG240M, the model parameters were: embedding dimension = 512, transformer heads = 8, transformer layers = 2, and dropout = 0.70. All models were trained using a learning rate of 10^{-4}, weight decay of 0.1, a batch size of 256, and the AdamW optimizer [35].

For the baseline experiments, the model was trained for 100 epochs. In the BLB experiments, training was split into 50 epochs for each model replica, followed by 50 epochs of fine-tuning for the averaged model, ensuring a consistent training

[1] https://github.com/AdityaVadlamani/BLB-HGNN.

budget for comparison. Unless stated otherwise, we fixed $s = 2$ and $b_{\text{rep}} = 0.5$ and ran each experiment with three random seeds.

Experiments with OGB_MAG were conducted on 2 A100 GPUs and 128 CPU cores, while experiments on MAG240M were run on 2 RTX A6000 GPUs and 16 CPU cores.

B. Results

The reported performance is the *average test micro-F1 score*, with the standard deviation indicated in parentheses (for tables) and depicted through shading (for figures). We report micro-F1 due to its popularity in node classification benchmarks like OGB(-LSC). For the MAG240M dataset, we also report the *test macro-F1 score* (see Figure 7) to demonstrate that the performance improvements are not solely due to the choice of metric.

Since MAG240M is part of OGB-LSC, to ensure a fair evaluation, the test set is hidden and cannot be self-evaluated. Therefore, for experiments with the MAG240M dataset, we modify the train/validation/test splits to be papers published *before 2018*, *during 2018*, and *during 2019*, respectively.

1) Baselines: We establish the baseline for assessing BLB−HGNN's effectiveness by comparing it against training a single model on the same reduced dataset, $\mathcal{D}_{\text{train}_0}$. We consider values of $b \in \{0.1, 0.2, 0.3, 0.4, 1.0\}$, where b controls the fraction of the training set used. The primary focus is on the regime where $b < 0.5$, as this reflects realistic constraints in large-scale systems where both feature memory and training runtime are limited. By operating in this low-resource regime, we aim to demonstrate that BLB−HGNN can serve as a practical substitute for standard training, achieving comparable or better performance with significantly fewer resources. For each setting, $\mathcal{D}_{\text{train}_0}$ is sampled uniformly from the original $\mathcal{D}_{\text{train}}$, and training is performed using Data Parallelism.

2) Summary: Figures 5 and 6 illustrate the average baseline performance and the best results using BLB−HGNN from the other experiments for both datasets. For both datasets, we observe that BLB−HGNN consistently outperforms simple training on the same amount of training data, with improvements reaching up to 5%. Additionally, when using less than 50% of the training nodes, BLB−HGNN achieves results within $1\% - 2\%$ of simple training using the full dataset.

Figure 8 and 9 show the training time for BLB−HGNN versus the baseline model for the fixed training budget of 100 epochs. We observe that the training time of BLB−HGNN is comparable to the baseline, so there is no additional cost in terms of training time with our method.

C. Ablation Study

1) Parameter Analysis: BLB−HGNN has two main hyperparameters: (1) the number of subsets/model replicas, s, and (2) the fraction of $\mathcal{D}_{\text{train}_0}$ to train a particular model replica on, b_{rep}. We perform three experiments to analyze these parameters for varying values of b.

In the first experiment, we set $s \in \{2, 3, 4\}$ and let $b_{\text{rep}} = 1/s$. Tables II and III show that a greater value of s, i.e., having

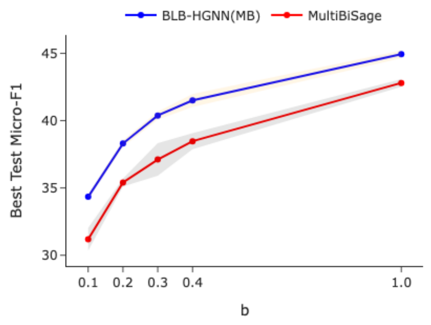

Fig. 5. MultiBiSage vs Best BLB−HGNN performance on OGB_MAG with the same amount of training data (MultiBiSage - MB).

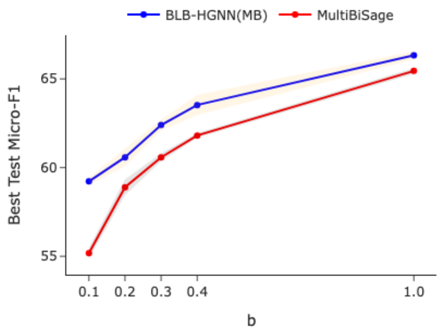

Fig. 6. MultiBiSage vs Best BLB−HGNN performance on MAG240M with the same amount of training data (MultiBiSage - MB).

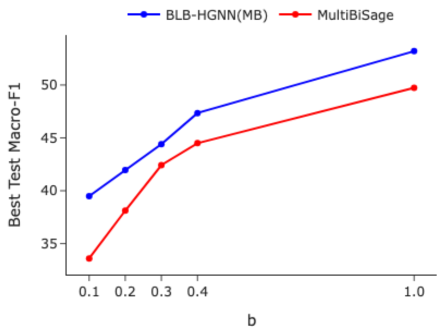

Fig. 7. MultiBiSage vs BLB−HGNN on MAG240M *macro-F1* performance with the same amount of training data (MultiBiSage - MB)

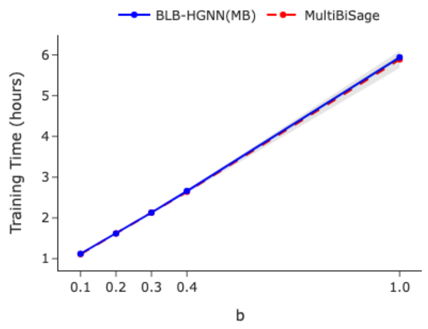

Fig. 8. Training time of MultiBiSage vs Best BLB−HGNN on OGB_MAG in hours (MultiBiSage - MB)

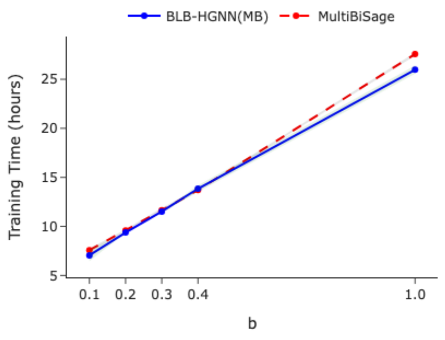

Fig. 9. Training time of MultiBiSage vs Best `BLB-HGNN` on MAG240M in hours (MultiBiSage - MB)

more models trained and then merged, doesn't necessarily imply a better final performance. We observe that only 2 replicas are enough to see the performance improvements. This observation holds for both MAG240M and OGB_MAG.

The second experiment has the same setup as the previous one; however, we assert the extra condition that model replicas are trained on distinct data, i.e., the $\mathcal{D}_{\text{train}_i}$ form a *partition* of $\mathcal{D}_{\text{train}_0}$. Table III shows that, like the original Bag of Little Bootstraps, we can choose whether the subsamples overlap or form a partition without sacrificing performance. This partitioned setup also aligns with federated or cross-device learning scenarios, where data privacy or storage constraints limit access to full datasets. The ability to train non-overlapping replicas and successfully merge them into a strong global model positions `BLB-HGNN` as a promising approach for federated HGNN training.

In the third experiment, we set $s \in \{3, 4\}$ and let $b_{\text{rep}} = 0.5 > 1/s$. Table IV shows that a larger b_{rep} improves performance. This is because a larger b_{rep} forces each model replica to see more unique data points, strengthening that replica before the model merging process.

In all three experiments, `BLB-HGNN` provides a performance improvement of up to 5% over the baselines.

TABLE II
Impact of subsampling fraction of $\mathcal{D}_{\text{TRAIN}}$ (b) and replica counts (s) with subsampling fraction per training replica (b_{REP}) set to $1/s$ on MAG240M (MultiBiSage)

$b\backslash s$	**2**	**3**	**4**
0.1	**59.22(±0.21)**	57.52(±1.58)	56.97(±1.24)
0.2	**60.58(±0.49)**	60.29(±1.34)	59.47(±0.94)
0.3	61.57(±0.32)	**62.40(±0.31)**	61.55(±0.16)
0.4	**63.53(±0.56)**	62.82(±0.89)	62.36(±0.63)
1.0	**66.33(±0.23)**	66.04(±0.19)	65.44(±0.22)

2) Model Merging: In Table V, we compare different methods – parameter averaging, parameter averaging with fine-tuning, logit averaging, and majority voting – for merging and ensembling the s trained model replicas in `BLB-HGNN` for both the OGB_MAG and MAG240M datasets. We observe

TABLE III
Impact of overlapping training data across varying subsampling fraction of $\mathcal{D}_{\text{TRAIN}}$ (b) and replica counts (s) with subsampling fraction per training replica (b_{REP}) set to $1/s$ on OGB_MAG (MultiBiSage)

$s = 2$		
$b\backslash$	Overlap	No Overlap
0.1	33.19(±0.50)	**33.55(±0.71)**
0.2	36.72(±0.16)	**37.67(±0.68)**
0.3	**38.86(±0.27)**	37.95(±0.46)
0.4	40.15(±0.70)	**40.20(±0.34)**
1.0	**44.93(±0.28)**	44.62(±0.72)
$s = 3$		
$b\backslash$	Overlap	No Overlap
0.1	32.78(±0.32)	**32.79(±0.91)**
0.2	**36.71(±0.62)**	36.62(±1.15)
0.3	**38.70(±0.49)**	38.48(±0.21)
0.4	**39.43(±0.65)**	39.14(±0.30)
1.0	43.56(±0.07)	**44.48(±0.65)**
$s = 4$		
$b\backslash$	Overlap	No Overlap
0.1	32.29(±0.77)	**32.91(±0.63)**
0.2	**36.95(±0.74)**	36.68(±0.30)
0.3	**38.08(±0.36)**	38.00(±0.26)
0.4	39.18(±0.38)	**39.32(±1.13)**
1.0	**43.30(±0.55)**	43.10(±0.41)

TABLE IV
Impact of subsampling fraction of $\mathcal{D}_{\text{TRAIN}}$ (b) and replica counts (s) with a fixed subsampling fraction per training replica ($b_{\text{REP}} = 1/2$) on OGB_MAG. (MultiBiSage)

$s = 3$		
$b\backslash b_{\text{rep}}$	$\mathbf{1/s}$	**0.50**
0.1	32.78(±0.32)	**32.85(±1.18)**
0.2	36.71(±0.62)	**37.12(±0.31)**
0.3	**38.70(±0.49)**	38.33(±0.67)
0.4	39.43(±0.65)	**40.22(±0.79)**
1.0	43.56(±0.07)	**44.36(±0.60)**
$s = 4$		
$b\backslash b_{\text{rep}}$	$\mathbf{1/s}$	**0.50**
0.1	32.29(±0.77)	**32.76(±0.71)**
0.2	**36.95(±0.74)**	36.65(±0.76)
0.3	38.08(±0.36)	**38.27(±0.90)**
0.4	39.18(±0.38)	**39.38(±0.66)**
1.0	43.30(±0.55)	**44.40(±0.22)**

that `BLB-HGNN`'s default of averaging parameters and fine-tuning *significantly* outperforms the other methods by at least 7% with similar or smaller deviation for all datasets and values of b. In some cases, the improvements are between 10% to 20%.

3) Sampling Methods: In Table VI, we compare several sampling strategies for constructing $\mathcal{D}_{\text{train}_0}$ on the OGB_MAG dataset. Beyond uniform sampling, we evaluate: (1) Personalized PageRank (PPR) with uniformly sampled seed nodes, (2) Spread Sampling, and (3) PPR with seed nodes selected via

TABLE V
EFFECT OF MODEL MERGING METHODS (MULTIBISAGE)

	MAG240M				OGB_MAG			
	Avg Params w/ fine-tuning	Avg Params	Majority Voting	Avg Logits	Avg Params w/ fine-tuning	Avg Params	Majority Voting	Avg Logits
0.1	$\mathbf{59.22}(\pm\mathbf{0.21})$	$23.60(\pm2.45)$	$36.77(\pm0.89)$	$41.70(\pm1.15)$	$\mathbf{33.19}(\pm\mathbf{0.50})$	$20.26(\pm4.46)$	$21.05(\pm0.62)$	$23.33(\pm0.46)$
0.2	$\mathbf{60.58}(\pm\mathbf{0.49})$	$25.89(\pm12.66)$	$42.97(\pm0.49)$	$47.59(\pm0.57)$	$\mathbf{36.72}(\pm\mathbf{0.16})$	$19.03(\pm4.60)$	$26.29(\pm0.08)$	$29.39(\pm0.20)$
0.3	$\mathbf{61.57}(\pm\mathbf{0.32})$	$24.42(\pm3.68)$	$44.85(\pm1.40)$	$49.19(\pm1.36)$	$\mathbf{38.86}(\pm\mathbf{0.27})$	$22.16(\pm4.15)$	$27.89(\pm0.54)$	$30.99(\pm0.34)$
0.4	$\mathbf{63.53}(\pm\mathbf{0.56})$	$22.41(\pm1.44)$	$47.34(\pm0.72)$	$51.91(\pm0.63)$	$\mathbf{40.15}(\pm\mathbf{0.70})$	$17.41(\pm1.68)$	$26.30(\pm0.43)$	$29.60(\pm0.66)$
1.0	$\mathbf{66.33}(\pm\mathbf{0.23})$	$11.78(\pm4.15)$	$54.06(\pm0.46)$	$58.52(\pm0.96)$	$\mathbf{44.93}(\pm\mathbf{0.28})$	$15.62(\pm4.34)$	$33.66(\pm0.50)$	$36.90(\pm0.23)$

Spread Sampling (PPR-SS). For PPR, we reuse the random walks employed by MultiBiSage.

Our results show that all non-uniform sampling methods outperform uniform sampling, with gains of up to 2%. Notably, these advanced sampling techniques are highly parallelizable and can be efficiently precomputed in distributed environments and stored for reuse. The results suggest that the non-uniform sampling may act as a form of *implicit regularization*. By selecting more structurally diverse or semantically meaningful subsets, techniques like Spread Sampling may steer the training toward flatter minima, which can generalize better, even under storage constraints.

An important observation is that these methods allow the use of smaller b values while still achieving better performance than uniform sampling with larger b. For instance, Spread Sampling with $b = 0.3$ outperforms uniform sampling at $b = 0.4$. Interestingly, the hybrid PPR-SS method consistently performs between PPR and Spread Sampling, suggesting potential as a balanced strategy. Further tuning and analysis of PPR-SS is left as future work.

These advanced sampling techniques exploit the graph's topology to select more informative and representative training nodes. As shown in Table VI, Spread Sampling tends to be more effective on large graphs by promoting broad coverage and capturing nodes from diverse communities. In contrast, PPR often excels on smaller graphs, where random walks can traverse the entire structure, yielding a meaningful set of seed nodes with fewer constraints, more easily.

Given our observation that specialized sampling can improve the performance of `BLB-HGNN`, an interesting direction for future work is the development of *adaptive* sampling strategies. Integrating such approaches could further enhance the performance gains seen with `BLB-HGNN` and allow it to sample more informative nodes throughout training.

D. Generalizability

To demonstrate the generalizability of our method, we applied `BLB-HGNN` to SeHGNN. We modified the code shared on the OGB_MAG leaderboard to train with `BLB-HGNN`. We ran the original version for 100 epochs and our modified version with a $50/50$ split for the replica training and fine-tuning. We used all the default parameters and a single stage of training. Table VII shows that `BLB-HGNN` matches or improves the baseline performance.

TABLE VI
EFFECT OF SAMPLING METHODS ON OGB_MAG (MULTIBISAGE)

	Uniform	PPR
0.1	$33.19(\pm0.50)$	$\mathbf{34.34}(\pm\mathbf{0.18})$
0.2	$36.72(\pm0.16)$	$\mathbf{38.30}(\pm\mathbf{0.25})$
0.3	$38.86(\pm0.27)$	$39.55(\pm0.97)$
0.4	$40.15(\pm0.66)$	$41.14(\pm0.33)$
	PPR-SS	Spread Sampling
0.1	$33.48(\pm0.97)$	$33.38(\pm0.06)$
0.2	$37.95(\pm0.58)$	$37.94(\pm0.59)$
0.3	$40.00(\pm0.25)$	$\mathbf{40.38}(\pm\mathbf{0.29})$
0.4	$41.44(\pm0.77)$	$\mathbf{41.50}(\pm\mathbf{0.48})$

TABLE VII
SEHGNN VS `BLB-HGNN` WITH THE SAME AMOUNT OF TRAINING DATA ON OGB_MAG (SEHGNN - SE)

	SeHGNN	`BLB-HGNN` (Se)	Change
0.1	$43.81(\pm0.05)$	$43.39(\pm0.09)$	-0.42%
0.2	$46.40(\pm0.53)$	$46.31(\pm0.29)$	-0.07%
0.3	$47.26(\pm0.32)$	$47.84(\pm0.15)$	+0.58%
0.4	$48.36(\pm0.01)$	$48.81(\pm0.25)$	+0.45%
0.5	$49.16(\pm0.44)$	$49.50(\pm0.43)$	+0.34%
0.6	$49.70(\pm0.23)$	$50.45(\pm0.12)$	+0.75%
1.0	$51.03(\pm0.15)$	$51.73(\pm0.06)$	+0.70%

V. RELATED WORK

1) Heterogeneous Graph Mining: Real-world data frequently appears as heterogeneous graphs containing various types of nodes and edges along with different unstructured contents, e.g., images and text. Numerous models have been developed to extract information from heterogeneous graphs. HGT is proposed for web-scale heterogeneous graphs, which utilize meta-relations and incorporate relative temporal encoding to capture dynamic dependencies. HAN [36] captures node-level and semantic-level importance by a hierarchical attention mechanism. These models facilitate machine learning tasks, including personalized recommendation, node classification, and link prediction. However, they require the entire graph and all associated features to be available on a single machine, making them storage-intensive.

2) Random Walk Based Heterogeneous GNNs: One effective strategy to reduce feature memory during training is precomputing node neighborhoods and caching the relevant features. Unlike many HGNNs that rely on on-the-fly sampling [2], [37], random walk-based methods allow for offline neighborhood construction, improving efficiency and scalability. These methods treat frequently visited nodes in a random walk as neighbors and prefetch their features for training. Notable examples include MultiBiSage, HetGNN, and RAW-GNN [38], demonstrating strong performance on large-scale heterogeneous graphs using this paradigm.

3) Model Merging: Model merging is employed across several applications to improve the performance and robustness of ML models. The Branch-Train-Merge [25] algorithm trains LLMs by avoiding multi-node synchronization, using multiple expert language models on distinct textual domains, and then merging them for application to new domains. Model averaging approaches for Stochastic Gradient Descent and Latent Dirichlet Allocation have been proposed to enhance Spark's MLlib by having each worker node independently update its local model, which is then aggregated by averaging at a central node, reducing communication overhead and improving convergence speed [39]. Adaptive Federated Averaging [40] algorithm iteratively constructs average models from local updates and identifies unreliable clients by comparing the similarity of the local models to the aggregated model.

4) Bootstrapping: Apart from BLB, other computationally efficient variants include Subsampled Double Bootstrap [41] and Distributed Bootstrap [42]. Previous applications of BLB to machine learning have been towards random forests [43], including BLB-gcForest [44].

5) Graph Coarsening: Graph coarsening aims to reduce the size of a graph by merging nodes while preserving its structural properties, thereby enabling more scalable GNN training. Methods like MILE [45], HeteroMILE [46], ConvMatch [47] and MLC-GCN [48] recursively compress the graph, perform learning on the coarsened version, and refine embeddings for the original graph. While effective, these approaches primarily alter the graph topology. `BLB-HGNN` operates orthogonally by reducing training instances through sampling, without modifying the graph structure, making it particularly suitable for feature memory-constrained settings.

VI. Conclusion

We presented `BLB-HGNN`, a novel training algorithm that addresses the feature memory bottleneck in large-scale heterogeneous graph neural networks. By leveraging the Bag of Little Bootstraps framework, `BLB-HGNN` enables scalable training using only a subset of the data, while preserving or improving model performance. Our method trains multiple model replicas on resampled subsets, merges them through parameter averaging, and fine-tunes the merged model, achieving a performance gain of up to 5% with the same data budget.

We demonstrated that `BLB-HGNN` is model-agnostic and compatible with various sampling strategies, including Personalized PageRank and Spread Sampling, which further improve performance over uniform sampling. Additionally, our experiments across multiple datasets and HGNN architectures confirm that `BLB-HGNN` achieves competitive performance with reduced training data and no additional runtime cost.

`BLB-HGNN` offers a practical alternative for training on large graphs, particularly in storage-constrained settings. Future directions include adaptive sampling, theoretical analysis, and extensions to federated or decentralized training.

Acknowledgments

The authors acknowledge support from National Science Foundation (NSF) grants MRI OAC-2018627 and SES-1949037, and the Ohio Supercomputing Center for experimentation resources. We would also like to acknowledge the AI-Edge Institute (NSF CNS-2112471). Any opinions and findings are those of the author(s) and do not necessarily reflect the views of the granting agencies.

References

[1] Y. Gu, Y. Sun, and J. Gao, "The co-evolution model for social network evolving and opinion migration," *In Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 175–184, 2017.

[2] W. Hamilton, Z. Ying, and J. Lekovec, "Inductive representation learning on large graphs," *In Advances in Neural Information Processing Systems*, pp. 1024–1034, 2017.

[3] D. Eppstein, M. S. Paterson, and F. F. Yao, "On nearest-neighbor graphs," *Discrete & Computational Geometry*, vol. 17, no. 3, pp. 263–282, 1997.

[4] 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*, ser. KDD '18. ACM, Jul 2018. [Online]. Available: http://dx.doi.org/10.1145/3219819.3219890

[5] A. Pfadler, H. Zhao, J. Wang, L. Wang, P. Huang, and D. L. Lee, "Billion-scale recommendation with heterogeneous side information at taobao," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. Dallas, TX: IEEE, Apr. 2020, pp. 1667–1676.

[6] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 1263–1272.

[7] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. S. Pande, "Moleculenet: A benchmark for molecular machine learning," *CoRR*, vol. abs/1703.00564, 2017. [Online]. Available: http://arxiv.org/abs/1703.00564

[8] D. K. Bhattacharyya and J. K. Kalita, *Network anomaly detection: A machine learning perspective*. Chapman and Hall/CRC, 2013.

[9] T. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[10] P. Veličković, G. Cucurull, A. Casanov, A. Romero, P. Liò, and Y. Bengio, "Graph attention network," *ICLR*, 2018.

[11] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 793–803. [Online]. Available: https://doi.org/10.1145/3292500.3330961

[12] Z. Zhu, X. Fan, X. Chu, and J. Bi, "Hgcn: Heterogeneous graph convolutional network-based deep learning model toward collective classification," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1161–1171, 2020.

[13] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," *The Web Conference*, pp. 2704–2710, 2020.

[14] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "Ogb-lsc: A large-scale challenge for machine learning on graphs," *arXiv preprint arXiv:2103.09430*, 2021.

[15] S. Gurukar, N. Pancha, A. Zhai, S. Kim, E. Hu, S. Parthasarathy, C. Rosenberg, and J. Leskovec, “Multibisage: A web-scale recommendation system using multiple bipartite graphs at pinterest,” *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 781–789, Dec. 2022.

[16] 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,” *arXiv preprint arXiv:2005.00687*, 2020.

[17] A. Khatua, V. S. Mailthody, B. Taleka, T. Ma, X. Song, and W.-m. Hwu, “Igb: Addressing the gaps in labeling, features, heterogeneity, and size of public graph datasets for deep learning research,” in *In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’23)*, ser. KDD ’23, 2023. [Online]. Available: https://arxiv.org/abs/2302.13522

[18] R. Andersen, K. Lang, and F. Chung, “Local graph partitioning using pagerank vectors,” in *2006 47th Annual IEEE Conference on Foundations of Computer Science*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2006, pp. 475–486. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/FOCS.2006.44

[19] Y. Wang, B. Bandyopadhyay, V. Patel, A. Chakrabarti, D. Sivakoff, and S. Parthasarathy, “Spread sampling and its applications on graphs,” in *Complex Networks and Their Applications VIII*, H. Cherifi, S. Gaito, J. F. Mendes, E. Moro, and L. M. Rocha, Eds. Cham: Springer International Publishing, 2020, pp. 128–140.

[20] A. Kleiner, A. Talwalker, P. Sarkar, and M. I. Jordan, “The big data bootstrap,” 2012.

[21] Y. Dong, N. V. Chawla, and A. Swami, “metapath2vec: Scalable representation learning for heterogeneous networks,” *KDD*, 2017.

[22] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard, “Complex embeddings for simple link prediction,” in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML’16. JMLR.org, 2016, p. 2071–2080.

[23] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[24] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, and L. Schmidt, “Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time,” 2022.

[25] M. Li, S. Gururangan, T. Dettmers, M. Lewis, T. Althoff, N. A. Smith, and L. Zettlemoyer, “Branch-train-merge: Embarrassingly parallel training of expert language models,” 2022.

[26] S. Sanyal, A. Neerkaje, J. Kaddour, A. Kumar, and S. Sanghavi, “Early weight averaging meets high learning rates for llm pre-training,” 2023.

[27] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. G. Wilson, “Averaging weights leads to wider optima and better generalization,” 2019.

[28] A. Ramé, N. Vieillard, L. Hussenot, R. Dadashi, G. Cideron, O. Bachem, and J. Ferret, “Warm: On the benefits of weight averaged reward models,” 2024.

[29] P. Moritz, R. Nishihara, and et al., “Ray: A distributed framework for emerging ai applications,” *arXiv preprint arXiv:1712.05889*, 2018.

[30] K. Wang, Z. Shen, C. Huang, C.-H. Wu, Y. Dong, and A. Kanakia, “Microsoft Academic Graph: When experts are not enough,” *Quantitative Science Studies*, vol. 1, no. 1, pp. 396–413, 02 2020. [Online]. Available: https://doi.org/10.1162/qss_a_00021

[31] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” *arXiv preprint arXiv:1903.02428*, 2019.

[32] Y. Shi, P. Team, Z. Huang, W. Li, W. Su, and S. Feng, “Runimp: Solution for kddcup 2021 mag240m-lsc.” 2021. [Online]. Available: https://ogb.stanford.edu/paper/kddcup2021/mag240m_BD-PGL.pdf

[33] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan, “Simple and efficient heterogeneous graph neural network,” in *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence*, ser. AAAI’23/IAAI’23/EAAI’23. AAAI Press, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i9.26283

[34] W. Falcon and The PyTorch Lightning team, “PyTorch Lightning,” Mar. 2019. [Online]. Available: https://github.com/Lightning-AI/lightning

[35] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *ICLR*, 2019.

[36] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, “Heterogeneous graph attention network,” in *The World Wide Web Conference*, ser. WWW ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2022–2032. [Online]. Available: https://doi.org/10.1145/3308558.3313562

[37] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, “Graphsaint: Graph sampling based inductive learning method,” in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=BJe8pkHFwS

[38] D. Jin, R. Wang, M. Ge, D. He, X. Li, W. Lin, and W. Zhang, “Raw-gnn: Random walk aggregation based graph neural network,” in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, L. D. Raedt, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2022, pp. 2108–2114, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2022/293

[39] Y. Guo, Z. Zhang, J. Jiawei, W. Wu, C. Zhang, B. Cui, and J. Li, “Model averaging in distributed machine learning: a case study with apache spark,” *The VLDB Journal*, vol. 30, pp. 1–20, 07 2021.

[40] L. Muñoz-González, K. T. Co, and E. C. Lupu, “Byzantine-robust federated machine learning through adaptive model averaging,” 2019.

[41] S. Sengupta, S. Volgushev, and X. Shao, “A subsampled double bootstrap for massive data,” 2015.

[42] Y. Yu, S.-K. Chao, and G. Cheng, “Simultaneous inference for massive data: Distributed bootstrap,” 2020.

[43] P. de Viña and G. Martínez-Muñoz, “Using bag-of-little bootstraps for efficient ensemble learning,” in *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27*. Springer, 2018, pp. 538–545.

[44] Z. Chen, T. Wang, H. Cai, S. K. Mondal, and J. P. Sahoo, “Blbgcforest: A high-performance distributed deep forest with adaptive subforest splitting,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 3141–3152, 2022.

[45] J. Liang, S. Gurukar, and S. Parthasarathy, “Mile: A multilevel framework for scalable graph embedding,” in *Proceedings of the Fifteenth International AAAI Conference on Web and Social Media (ICWSM)*. AAAI Press, 2021. [Online]. Available: https://github.com/jiongqian/MILE

[46] Y. Zhang, Y. He, S. Gurukar, and S. Parthasarathy, “Heteromile: a multi-level graph representation learning framework for heterogeneous graphs,” *arXiv preprint arXiv:2404.00816*, 2024.

[47] C. Dickens, E. Huang, A. Reganti, J. Zhu, K. Subbian, and D. Koutra, “Graph coarsening via convolution matching for scalable graph neural network training,” in *Companion Proceedings of the ACM Web Conference 2024*. ACM, 2024, pp. 1502–1511. [Online]. Available: https://doi.org/10.1145/3589335.3651920

[48] Y. Xie, C. Yao, M. Gong, C. Chen, and A. K. Qin, “Graph convolutional networks with multi-level coarsening for graph classification,” *Knowledge-Based Systems*, vol. 194, p. 105578, 2020. [Online]. Available: https://doi.org/10.1016/j.knosys.2020.105578

Appendix

Symbols Reference

We provide a full list of referenced symbols in Table VIII.

TABLE VIII
Notation of the BLB-HGNN Algorithm

Symbol	**Description**
$\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{test}}$	Training, validation, and test datasets
b	Subsampling fraction for $\mathcal{D}_{\text{train}}$
b_{rep}	Subsampling fraction per bootstrap replica
s	Number of model replicas (ensemble size)
r	Number of training epochs per replica
r_{avg}	Number of epochs to fine-tune the averaged model
$\mathcal{M}_i$	The i-th model replica
$\mathcal{M}_{\text{avg}}$	Averaged model parameters across s replicas
$\mathcal{D}_{\text{train}_0}$	Initial subsampled training dataset
$\mathcal{D}_{\text{train}_i}$	Subsampled dataset for the i-th replica
$\mathcal{D}_{\text{blb}}$	Bootstrapped dataset used in BLB training