2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) | 978-1-7281-8643-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ISPASS51385.2021.00029

Performance Analysis of Graph Neural Network Frameworks

Junwei Wu, Jingwei Sun, Hao Sun, Guangzhong Sun
University of Science and Technology of China, China
{wjw1996, ustcsh}@mail.ustc.edu.com, {sunjw, gzsun}@ustc.edu.com

***Abstract*—Graph neural networks (GNNs) are effective models to address learning problems on graphs and have been successfully applied to numerous domains. To improve the productivity of implementing GNNs, various GNN programming frameworks have been developed. Both the effectiveness (accuracy, loss, etc) and the performance (latency, bandwidth, etc) are essential metrics to evaluate the implementation of GNNs. There are many comparative studies related to the effectiveness of different GNN models on domain tasks. However, the performance characteristics of different GNN frameworks are still lacking.**

In this study, we evaluate the effectiveness and performance of six popular GNN models, GCN, GIN, GAT, GraphSAGE, MoNet, and GatedGCN, across several common benchmarks under two popular GNN frameworks, PyTorch Geometric and Deep Graph Library. We analyze the training time, GPU utilization, and memory usage of different evaluation settings and the performance of models across different hardware configurations under the two frameworks. Our evaluation provides in-depth observations of performance bottlenecks of GNNs and the performance differences between the two popular GNN frameworks. Our work helps GNN researchers understand the performance differences of the popular GNN frameworks, and gives guidelines for developers to find potential performance bugs of frameworks and optimization possibilities of GNNs.

***Index Terms*—Graph neural network, GNN framework, Performance evaluation**

I. Introduction

Graphs are a type of data structure that models a set of objects (nodes) and their relationships (edges). In recent years, researches on analyzing graphs with machine learning attract much attention. Graphs have great expressive power, thus they can denote a large number of complex systems across various domains, such as social science (social networks) [1], [2], natural science (physical systems [3], [4] and protein-protein interaction networks [5]), knowledge graphs [6] and combinatorial optimization algorithms [7]. As a unique non-Euclidean data structure for machine learning, graphs can be adopted to node classification [2], link prediction [8], and clustering [9]. Graph neural networks (GNNs) [10] [11] [12] are deep learning based methods that operate on graph domain. Due to its convincing effectiveness and high interpretability, GNN has been a widely applied graph analysis method.

With the development of various GNN models, to facilitate the implementation of models, several GNN frameworks have been proposed. PyTorch Geometric (PyG) [13] and Deep Graph Library (DGL) [14] are the current mainstream GNN frameworks. These frameworks provide convenient programming interfaces and power libraries of operators and layers proposed in the frontier of GNN studies. The GNN models based on these frameworks can usually achieve better training time performance than that based on general-purpose deep learning frameworks.

In consideration of the existence of various GNN frameworks, it is necessary to evaluate the performance of these frameworks in the implementation of GNN models. Currently, the existing studies related to GNN models mainly focus on the effectiveness of different GNN models on domain tasks. And there are few works that put attention on the performance characteristics of GNN frameworks. In this study, we perform a performance analysis and comparison of how these different applications behave on popular GNN frameworks and obtain useful insights.

We present a performance evaluation of six GNN models: GCN [2], GIN [15], GAT [16], GraphSAGE [17], MoNet [18] and GatedGCN [19], on five datasets: Cora [20], PubMed [20], ENZYMES [21], DD [21] and MNIST [18], in two GNN frameworks: PyTorch Geometric and Deep Graph Library.

In our evaluation, models in the two frameworks have similar statistical test accuracy but the training time performance of models in DGL is significantly worse than that of PyG. In general, the memory usage of models in DGL is more than that in PyG. For anisotropic models such as GAT and GatedGCN, memory usage is quite high on relatively large datasets which may need new careful processing for memory usage. On datasets with a large number of nodes and long node features, GNN models do not make good use of the parallelism of GPU training. In GNNs, the inter-node and intra-node parallelism of GNNs deserves further study. The data loading time for tasks of graph level is a major part of training time because batching multiple graphs into a single large graph is pretty time-consuming, which also leads to the low GPU utilization of GNNs in the two frameworks.

In summary, we make the following contributions.

- We show the performance characteristics of the six GNN models under the two frameworks on node classification tasks and graph classification tasks in a single GPU context.
- We analyze the hardware resource usage of the six models under the two frameworks. We show the GPU utilization and peak memory usage of training the models in the two frameworks and provide performance bugs and optimization possibilities of GNNs on hardware resource usage in the two frameworks.

978-1-7281-8643-6/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPASS51385.2021.00029

- We analyze the training time performance of training the models in the two frameworks in single-GPU and multi-GPU contexts. We present different parts of training time and detailed layer-wise execution time of the six models in the two frameworks. We analyze differences in training time performance of models under the two frameworks, identify performance limiting factors of models training in the two frameworks, and provide optimization possibilities of GNNs.

II. Backgound and Related Work

A. Graph Neural Networks

An increasing number of machine learning tasks require dealing with large graph datasets, which capture rich and complex relationships among potentially billions of elements. Graph Neural Network (GNN) becomes an effective way to address the graph learning problem by converting the graph data into a low dimensional space while keeping both the structural and property information to the maximum extent and constructing a neural network for training and referencing.

GNNs have been explored in a wide range of problem domains across supervised, semi-supervised, unsupervised, and reinforcement learning settings. The applications [11] can be divided into the three scenarios: (1) Structural scenarios where the data has explicit relational structure, such as physical systems, molecular structures, and knowledge graphs; (2) Non-structural scenarios where the relational structure is not explicit include image, text, etc; (3) Other application scenarios such as generative models and combinatorial optimization problems.

GNNs can be categorized into five groups: graph convolutional networks, graph attention networks, graph auto-encoders, graph generative networks, and graph spatial-temporal networks. These GNNs adopt different types of message delivery methods or different mechanisms, which can be used for different types of graphs. Encouraged by the success of convolutional neural networks [22], graph convolutional networks re-define the notion of convolution for graph data to achieve propagating neighbor information. Graph attention networks introduce attention mechanisms, in which more important nodes are given more weight. Graph auto-encoders are unsupervised learning frameworks, aiming at representing network nodes as low-dimensional vector representations through the encoder, and then reconstruct the graph data through the decoder.

In Graph Convolutional Networks (GCN) [2], the propagation rule can be recovered by the formula:

$$h_i^{k+1} = \sigma(W^k \frac{1}{deg_i} \sum_{j \in N_i} h_j^k), \tag{1}$$

where h_i^k is a vector of activations of node i in the k^th neural network layer, N_i is the set of nodes connected to node i on the graph, $deg_i = |N_i|$ is the degree of node i, $\sigma(\cdot)$ denotes a differentiable, non-linear activation function, and W^k is a layer-specific weight matrix.

Based on GCN, in the mean version of GraphSage [17], the equation in (1) is replaced with

$$h_i^{k+1} = \sigma\big(W^k Concat(h_i^{k+1}, \frac{1}{deg_i} \sum_{j \in N_i} h_j^k)\big), \tag{2}$$

and the embeddings vectors are projected onto the unit ball before being passed to the next layer. In Graph Isomorphism Network (GIN) [15], node representations are updated as

$$h_i^{k+1} = \sigma\left(W^k \sigma\left(BN\left(V^k\big((1+\epsilon^k) + \sum_{j \in N_i} h_j^k\big)\right)\right)\right), \tag{3}$$

where ϵ^k, W^k, V^k are learnable parameters and BN is the Batch Normalization layer [23]. In all the above models, each neighbor contributes equally to the update of the central node. these model are refered to as isotropic that they treat every "edge direction" equally.

Graph Attention Networks (GAT) [16], Gaussian Mixture Model Networks (MoNet) [18] and Graph Convolutional Networks (GatedGCN) [19] propose anisotropic update schemes of the type

$$h_i^{\delta+1} = w_i^k h_i^k + \sum_{j \in N_i} w_{ij}^k h_j^k, \tag{4}$$

where the weights w_i^k and w_{ij}^k are computed using various mechanisms (e.g. attention mechanism in GAT or gating mechanism in GatedGCN).

B. GNN Frameworks and Low-level Libraries

Frameworks and low-level libraries are designed to simplify the life of programmers and to help them efficiently utilize existing complex hardware. Most GNN frameworks (e.g., PyTorch Geometric or Deep Graph Library) are based on currently popular DNN frameworks (e.g., PyTorch [24], MXNet [25] or TensorFlow [26]), which usually provide users with portable APIs to define computation logic, transforming the user program into an internal intermediate representation (e.g., dataflow graph representation [25]), and then become the basis for backend execution, including data transfers, memory allocations, and low-level CPU function calls or GPU kernel invocations. Such low-level functions are usually provided by libraries such as cuDNN [27] and cuBLAS [28], which provide efficient implementations of basic vector and multi-dimensional matrix operations (some operations are NN-specific such as convolutions or poolings) in C/C++ (CPU) or CUDA (GPU).

C. Related Work

Existing empirical evaluations about GNNs mainly focus on the accuracy of different GNN models. Shchur et al. [29] implemented GCN, MoNet, GAT, and GraphSage within the framework they proposed in their work. The authors performed a thorough experimental evaluation of the four GNN workloads on the transductive semi-supervised node classification task.

Errica et al. [30] performed a large number of experiments on chemical datasets and social datasets within a rigorous

model selection and assessment framework, in which all models (DGCNN, DiffPool, ECC, GIN, and GraphSage) were compared using the same features and the same data splits. The models in the work are implemented in PyG.

Dwivedi et al. [31] released an open benchmark infrastructure for GNNs. In the benchmark infrastructure, The authors went beyond the popular but small CORA and TU datasets and introduced medium-scale datasets. The authors evaluated six GNN workloads on their proposed benchmark datasets and identified important building blocks of GNNs (graph convolutions, anisotropic diffusion, residual connections, and normalization layers). The implementation of models in their work is base on DGL.

Mingyu Yan et al. [32] characterized the execution pattern of three GCN models (GCN, GIN, and GraphSage) on GPUs and proposed several useful guidelines for both software optimization and hardware optimization. The authors analyzed the aggregation phase, combination phase, and overall execution of training the three models on several datasets with node classification tasks. The implementation of models is based on PyG.

III. Evaluation Methodology

A. GNN models

To understand and compare the performance pattern of GNNs training workloads across the two GNN frameworks, we choose six typical models (GCN, GIN, GraphSAGE, GAT, MoNet, and GatedGCN) with different structures. The first three models belong to isotropic models and The latter three models belong to anisotropic models.

B. Frameworks and Tools

There are many open-source GNN frameworks or libraries, such as Graph Nets library [33], PyTorch Geometric, Deep Graph Library, GraphVite [34], Angel [35] and AliGraph [36]. Currently, the GNN frameworks widely used by researchers are PyTorch Geometric and Deep Graph Library. Almost all works that involve implementing models using the GNN framework now choose to use PyG or DGL. Compared to other frameworks, PyG and DGL show good performance advantages and provide more complete and convenient programming interfaces for GNN models implementation. The implementation of GNN frameworks usually bases on existing DNN frameworks. PyG only supports PyTorch as the backend and DGL supports PyTorch, TensorFlow, and MXNet as the backend now.

In our evaluations, the two frameworks, PyG and DGL, all use PyTorch as the backend. We implement the experimental code with python 3.7, PyTorch 1.5.0, DGL 0.5.1, PyG 1.6.1, and CUDA 10.1, running on CentOS 7.4 system. The frameworks are based on cuDNN 7.6.3. We collect the profiling data using the Nvprof profiler and Nsight Compute [37]. We use the NVIDIA System Management Interface Nvidia-smi to monitor the memory usage of GPUs.

TABLE I
Datasets Statiscs

Dataset	Cora	PubMed	ENZYMES	MNIST	DD
#Graph	1	1	600	70000	1178
#Nodes(Avg.)	2708	19717	32.63	70.57	284.32
#Edges(Avg.)	5429	44338	62.14	564.53	715.66
#Feature	1433	500	18	1	89
#Classes	7	3	6	10	2

TABLE II
Hyperparameter settings for node classification experiments.

Model	**Hyperparameter**		
	hidden	***lr***	***Other***
GCN	80	0.01	readout:mean
GAT	32	0.01	n_heads:8;readout:mean
GIN	64	0.005	neighbor_aggr_GIN:sum; readout:mean
GraphSage	32	0.001	sage_aggregator:mean_pool; readout:mean
MoNet	64	0.003	kernel:2; pseudo_dim_MoNet:2; readout:mean
GatedGCN	64	0.001	readout:mean

C. Workloads and Datasets

For our evaluations, we choose six popular graph neural networks used for node classification and graph classification across the two GNN frameworks. For the task of node classification, we select the datasets Cora and PubMed, which are widely used in the research field of graph neural network. In the citation network datasets — Cora and PubMed [20] (Collective classification in network data), nodes are documents and edges are citation links. For the task of graph classification, we selected the datasets ENZYMES, DD, and MNIST. The original MNIST is a popular image classification dataset from computer vision and is converted to graphs using super-pixels in our work. Super-pixels represent small regions of homogeneous intensity in images and can be extracted with the SLIC technique [38]. Dataset statistics used in our work are summarized in Table I. In each task, the amount of data in the datasets we selected is significantly different, which can reflect the impact of the scale of datasets on the performance of the models.

In this paper, we compare the following six popular graph neural network architectures. GCN is their simplest form that works by performing a linear approximation to spectral graph convolutions. MoNet generalizes the GCN architecture and allows learning adaptive convolution filters. The authors of GAT propose an attention mechanism that allows weighing nodes in the neighborhood differently during the aggregation step. GraphSAGE is a comprehensive improvement of the original GCN. GraphSAGE replaced full graph Laplacian with learnable aggregation functions, which are key to perform message passing and generalize to unseen nodes. The GIN architecture is based on the Weisfeiler-Lehman Isomorphism Test [39] to study the expressive power of GNNs. GatedGCN considers residual connections, batch normalization, and edge

gates to design another anisotropic variant of GCN.

Implementations of the same models on different frameworks might vary in a few aspects that can impact performance profiling results. Different implementations might have different hard-coded values for key hyper-parameters (e.g., learning rate, momentum, dropout rate, weight decay) in their code. To identify framework-specific performance characteristics, rather than just implementation-specific details, we adopt implementations of the same model to make them comparable across frameworks. We also ensure that they define the same network, i.e., the same types and sizes of corresponding layers and layers are connected in the same way. Moreover, we make sure that the key properties of the training algorithm are the same across implementations. We use the same optimizer (Adam [40]), same initialization, same learning rate decay, same maximum number of training epochs, early stopping criterion, patience, and validation frequency (display step) for models in different frameworks.

The hyperparameter settings for experiments in the node classification tasks and graph classification tasks are listed in Table II and Table III, respectively. Some hyperparameter settings that are not listed in the tables are described in section IV. In the tables, L is the number of layers. $hidden$ and out are the number of hidden and output features respectively. lr is the learning rate. $init\ lr$ is the initial learning rate. $patience$ is the decay patience. $min\ lr$ is the stopping learning rate.

IV. Evaluation

In this section, we present our evaluations and analysis of the training of six GNN workloads across two GNN frameworks using NVIDIA's 2080Ti GPUs.

A. Evaluation with Node Classification Tasks

In this subsection, we compare the training time and test accuracy for the GNN workloads across two GNN frameworks. We select two datasets: Cora (140 train, 500 validation, 1000 test nodes), PubMed (60 train, 500 validation, 1000 test nodes). Considering the small size of Cora and PubMed datasets, in all cases, we used full-batch training (using all nodes in the training set every epoch).

Setup: We keep the model architectures as what they are in the original papers or reference implementations. This includes the type and sequence of layers, choices of activation functions, and readout. We also fixed the number of attention heads for GAT to 8 and the number of Gaussian kernels for MoNet to 2, as proposed in the respective papers. All the models have 2 layers (input features $\rightarrow$ hidden layer $\rightarrow$ output layer). Because of the simple architecture of GNN models currently, the models can be well trained quickly. The maximum epoch times for all models are set to 200.

Results for node classification on Cora and PubMed are presented in Table IV. The table shows the time cost per epoch, total training time, classification accuracy, and standard deviation for six different GNN models implemented in two frameworks respectively.

Observations:

1) For node classification tasks, from our experimental results, we can find that the implementations with framework PyG, can get the best training time performance for all models. GAT gets the best test performance on Cora and GCN gets the best test performance on PubMed. In terms of test accuracy on Cora and PubMed, it is hard to tell the best between the two frameworks.

2) In general, the time per epoch of anisotropic models (GAT, MoNet, and GatedGCN) is longer than isotropic models (GCN, GIN, and GraphSage) because they employ complex mechanisms (such as sparse attention mechanism for GAT, and edge gates for GatedGCN).

3) Strangely, the training time performance of GatedGCN under DGL can be half of that under PyG. In DGL, we have to set the edge types parameter of GatedGCN although the dataset does not have this characteristic and then the features of all edge will be updated through a fully connected layer. The training time of GatedGCN under DGL is mainly spent on the edge feature update operation. In PyG, GatedGCN does not have the same operations on edges and the test performance is not reduced or even better compared to that of DGL.

B. Comparison with Graph Classification Tasks

In this section, we compare the training time and test accuracy for the GNN workloads across the two GNN frameworks. We select two datasets: ENZYMES (480 train, 60 validation, 60 test graphs, of sizes 2-126 nodes), DD (941 train, 118 validation, 119 test graphs, of sizes 30-5748 nodes).

We adopt the following experiment setup.

1) Splitting: For ENZYMES and DD, we perform a 10-fold cross-validation split, which gives 10 sets of train, validation, and test data indices in the ratio 8:1:1. We use stratified sampling to ensure that the class distribution remains the same across splits. The indices are saved and used across all experiments for fair comparisons.

2) Training: We use Adam optimizer with a learning rate decay strategy. An initial learning rate is tuned from a range of 1e-3 to 7e-5 for every GNN models. The learning rate is reduced by half, i.e., reduce factor 0.5, if the validation loss does not decrease after 25 epochs. The training stops when the learning rate decays to a value of 1e-6 or less. The model parameters at the end of training are used for evaluations on test sets. We use mini-batch training and the batch size is 128.

3) Accuracy: We use classification accuracy between the predicted labels and ground truth labels as our evaluation metric. Model performance is evaluated on the test split for all datasets. The reported performance is the average and standard deviation over all the 10 folds.

4) Graph classifier layer: We use a graph classifier layer which first builds a graph representation by averaging all node features extracted from the last GNN layer and then passing this graph representation to an MLP.

Our numerical results are presented in Table V. The table shows the time cost per epoch, total training time, classification accuracy, and standard deviation for six different GNN models implemented in the two frameworks respectively.

TABLE III
HYPERPARAMETER SETTINGS FOR GRAPH CLASSIFICATION EXPERIMENTS.

Models	Hyperparameter						
	L	hidden	out	Other	Learning Setup		
					Init lr	patience	min lr
GCN	4	128	128	readout: mean	7e-4	25	1e-6
GAT	4	32	256	n_heads: 8; readout: mean	1e-3	25	1e-6
GIN	4	80	80	neighbor_aggr_GIN: sum; learn_eps_GIN: Ture; neighbor_aggr_GIN: sum; readout: mean	7e-3	25	1e-6
GraphSage	4	96	96	sage_aggregator: meanpool; readout: mean	7e-4	25	1e-6
MoNet	4	80	80	kernel: 2; pseudo_dim_MoNet: 2; readout: mean	1e-3	25	1e-6
GatedGCN	4	96	96	edge_feat: False; readout: mean	7e-4	25	1e-6

TABLE IV
PERFORMANCE OF NODE CLASSIFICATION TASKS ON THE STANDARD TEST SETS OF CORA AND PUBMED

Dataset	Model	Framework			
		PyG		DGL	
		Epoch/Total	Acc±s.d.	Epoch/Total	Acc±s.d.
CORA	GCN	0.0049s/5.82s	80.8±1.3	0.0063s/8.21s	80.9±0.8
	GAT	0.0072s/6.24s	82.5±0.9	0.0082s/7.72s	81.7±1.1
	SAGE	0.0038s/5.33s	81.3±0.6	0.0068s/8.34s	81.1±0.9
	GIN	0.0058s/5.90s	75.8±1.4	0.0061s/8.09s	74.9±1.8
	MoNet	0.0068s/5.97s	79.7±1.6	0.0086s/8.19s	80.4±1.3
	GatedGCN	0.0054s/5.95s	76.8±1.9	0.0101s/8.74s	78.4±1.5
PubMed	GCN	0.0053s/6.36s	79.3±0.8	0.0071s/9.57s	78.7±0.9
	GAT	0.0082s/6.85s	77.9±0.7	0.0092s/10.9s	78.5±0.4
	SAGE	0.0050s/6.25s	77.6±0.9	0.0063s/9.45s	77.8±1.3
	GIN	0.0070s/6.52s	74.3±1.3	0.0079s/9.89s	73.8±1.7
	MoNet	0.0079s/6.74s	76.6±0.8	0.0094s/9.78s	77.8±0.8
	GatedGCN	0.0063s/6.34s	77.3±0.6	0.0174s/13.3s	74.5±1.2

TABLE V
PERFORMANCE OF GRAPH CLASSIFICATION TASKS ON THE STANDARD TEST SETS OF ENZYMES AND DD

Dataset	Model	Framework			
		PyG		DGL	
		Epoch/Total	Acc±s.d.	Epoch/Total	Acc±s.d.
ENZYMES	GCN	0.087s/249.3s	65.2±2.8	0.164s/416.3s	66.1±3.4
	GAT	0.117s/294.1s	66.7±3.1	0.195s/479.4s	64.3±2.6
	SAGE	0.071s/203.9s	67.8±3.0	0.157s/388.9s	66.1±3.8
	GIN	0.082s/223.5s	68.0±2.3	0.155s/387.5s	66.2±3.1
	MoNet	0.123s/308.6s	71.9±3.5	0.196s/477.1s	68.2±2.9
	GatedGCN	0.104s/273.3s	65.2±2.8	0.216s/528.5s	66.3±3.3
DD	GCN	0.361s/0.23hr	77.8±2.4	0.853s/0.54hr	78.6±1.7
	GAT	0.627s/0.38hr	76.2±2.8	1.042s/0.64hr	75.3±2.6
	SAGE	0.262s/0.17hr	74.2±2.6	0.603s/0.38hr	77.2±1.9
	GIN	0.484s/0.30hr	72.9±2.2	0.882s/0.53hr	77.9±2.6
	MoNet	0.434s/0.27hr	77.1±3.4	0.758s/0.47hr	75.4±3.8
	GatedGCN	0.355s/0.23hr	73.4±2.9	1.255s/0.77hr	71.6±3.1

Observations:

1) On the two datasets, most of the models implemented in the two frameworks respectively have similar statistical test accuracy. In terms of test accuracy, it is hard to tell the best between the two frameworks. on dataset ENZYMES, MoNet has the best test performance under both frameworks; On dataset DD, GCN gets the best test performance across both frameworks.

2) In general, anisotropic GNNs spend more training time but they may not get better test performance. The training time performance of the six models on the two datasets in PyG is significantly better than that in DGL. And the training time performance of GatedGCN under DGL is significantly worse than other models across the two frameworks.

C. Training Time Performance Analysis

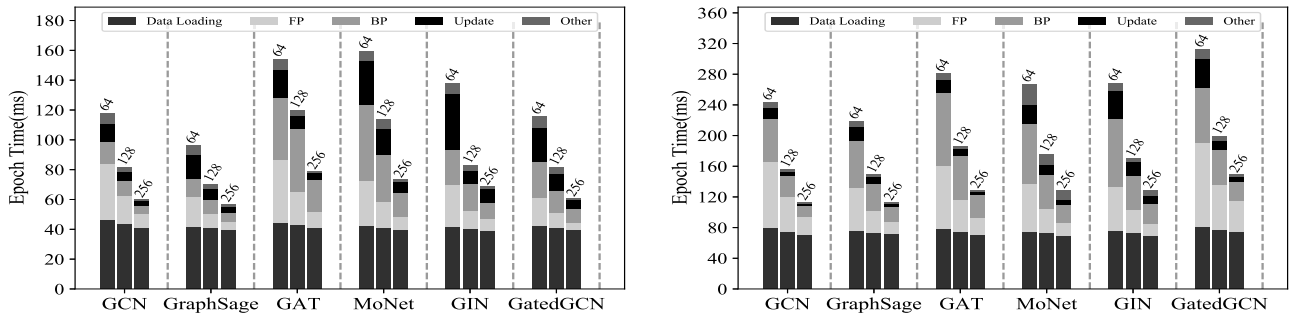
In this section, we look deeply into the training time of the six models across two GNN frameworks. We present the breakdown of execution time for one epoch and time for forward and backward propagation of training a batch of data. We carry out the experiments on dataset ENZYMES and DD, and the experiment settings are the same as our previous graph classification experiments on ENZYMES and DD.

For the breakdown of execution time per epoch, We break down the time into data loading time, forward propagation time, backward propagation time, parameters updating time, and other time under different batch sizes (64, 128, and 256).

Fig. 1 and Fig. 2 show the breakdown of execution time per epoch of six models across two frameworks training on datasets ENZYMES and DD, respectively. There are obvious differences between dataset ENZYMES and DD. Although the number of graphs in the DD dataset is only about twice as large as that in the ENZYMES dataset, the average number of nodes and edges of graphs and the dim of node features of DD are significantly larger than that of ENZYMES. Experiments on these two datasets can show the training time performance differences of GNN models on different scale datasets under the two frameworks.

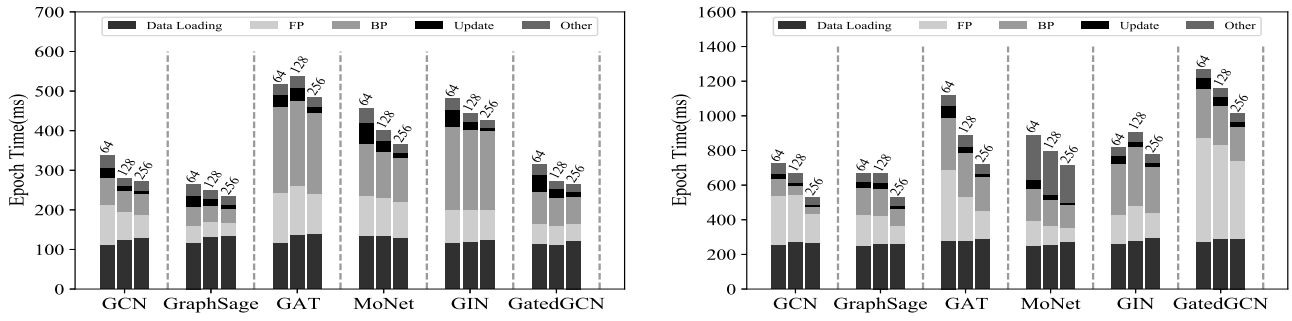
Data loading time of training GNN models on ENZYMES and DD takes up a large proportion of total training time. Data loading time includes not only data fetching from memory, but also data processing, and the data processing time is the main part of data loading time. The data processing operation models a batch of graphs as one big and disconnected graph. Data processing realizes that when the big graph is training, multiple graphs are training in fact and the node features in each graph are updated. And because there are no edges between each original graph in the big graph, the training of each original graph will not be affected by each other.

In Fig.1 and Fig.2, the data loading time of DGL is significantly longer than that of PyG across all models which mainly results from the time cost of data processing. Compared to the PyG, the implementation of data processing in DGL considers



(a) PyG (b) DGL

Fig. 1. Executiom time breakdown for six models with different batch size across two frameworks on ENZYMES.



(a) PyG (b) DGL

Fig. 2. Execution time breakdown for six models with different batch size across two frameworks on DD.

the type of nodes and edges which means heterogeneous graphs are supported in DGL. Although graphs in dataset ENZYMES and DD are not heterogeneous graphs, all graphs are treated as heterogeneous graphs during data processing, which brings extra-time loss. What is more, because DGL supports different DNN frameworks as backend, the data processing in DGL is not based on the PyTorch framework and can not use the highly efficient data operations provided by PyTorch. Besides, PyG proposed an advanced mini-batching strategy [13] in which there is no computational or memory overhead.

In Fig.1, on dataset ENZYMES, as the batch size doubled, the forward and backward propagation time nearly halved for all models across two frameworks. However, in Fig.2, on dataset DD, as the batch size doubled, the forward and backward propagation time is only slightly less or even larger for models across two frameworks. Compared to graphs in data ENZYMES, graphs in dataset DD have more nodes and edges, and the dim of node feature is larger. On datasets with a large number of nodes and edges, GNN models do not make good use of the parallelism of GPU training.

For the six GNN models selected in our experiments, the time cost of models in DGL is significantly higher than that in PyG. The execution time of training the six models on ENZYMES with a batch of input graphs across two frameworks is presented in Fig.3 and the batch size is 128. Compared to PyG, the conv layers of all models provided by DGL are more time-consuming.

Key operations of graph neural networks are to aggregate messages of neighbor nodes and update their nodes. In the implementation of these models in PyG and DGL, the key operations may not take up most of the training time. For example, in the implementation of GCN, the time for normalizing node features is longer than the time for key operations, besides, the node features are normalized before and after updating by the key operations which mainly results in the differences in GCN training time between DGL and PyG. In addition, although the training time of GAT in DGL is longer than that in PyG, the time cost for key operations in DGL is smaller than that in PyG. And this is because computing attention parameters for GAT in DGL takes more time than PyG. Compared to PyG, the conv layer of GNN models in DGL mostly uses more operations. For example, in GAT and GatedGCN, the edge features or attention are updated under DGL.

In DGL, conv1 for the six models takes more time than the other three conv layer after it. In conv1, the GSpMM (Generalized Sparse-Matrix Dense-Matrix Multiplication) functions that implement the key operations are more time-consuming than that in other conv layers for some models such as GIN. The GSpMM fuses two steps, computing messages by the source node and edge features and aggregating the messages

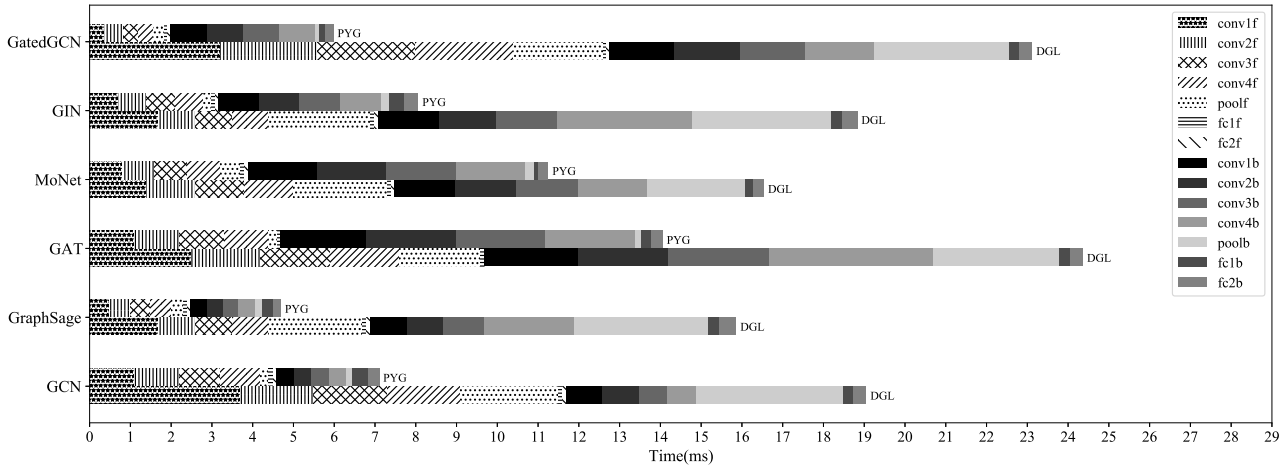


Fig. 3. The execution time of training six GNN models on ENZYMES with a batch of input graphs across two frameworks.

as the features on destination nodes, into one kernel. In some other models, such as GAT, the features and attention computing is more time-consuming in conv1.

The pooling operations provided by DGL, which get the features of graphs by the features of nodes in the graphs, are also more time-consuming than those provided by PyG. In PyG, the pooling operations are based on the scatter API of PyTorch. And in DGL, the pooling operation is based on their segment reduction operator.

D. Memory Usage and GPU Utilization Analysis

In this subsection, we evaluate and analyze the memory usage and GPU utilization in our previous graph classification experiment on dataset ENZYMES and DD. In the experiments, we use mini-batch training and evaluate the peak memory usage and GPU utilization with different batch sizes (64, 128, and 256).

GPU Compute Utilization. The GPU is the workhorse behind GNN training, as it is the unit responsible for executing the key operations involved in GNN training (break down into basic operations, such as vector and matrix operations). Therefore, for optimal throughput, the GPU should be busy all the time. Low utilization indicates that throughput is limited by other resources, such as CPU or data communication, and further improvement can be achieved by overlapping CPU runtime or data communication with GPU execution.

We define GPU Compute Utilization as the fraction of time that the GPU is busy (i.e. at least one CUDA core is active). For the training time period, it reports what percentage of time one or more GPU kernel(s) was active.

$$GPU\ utilization = \frac{GPU\ active\ time \times 100}{total\ elapsed\ time}\% \quad (5)$$

Peak Memory Usage. In the field of deep learning, when training DNNs, in addition to compute cycles, the physical memory capacity has become a performance bound. Currently, the main work in the field of graph neural network is to propose or improve GNN models and study the applications of GNNs. There are few works in the field that have paid attention to memory usage in the GNNs training process. We measure the peak memory usage of models across frameworks during the training process.

Fig. 4 and Fig. 5 show the peak memory usage and GPU compute utilization of six models training on datasets ENZYMES and DD across two frameworks, respectively.

Observations:

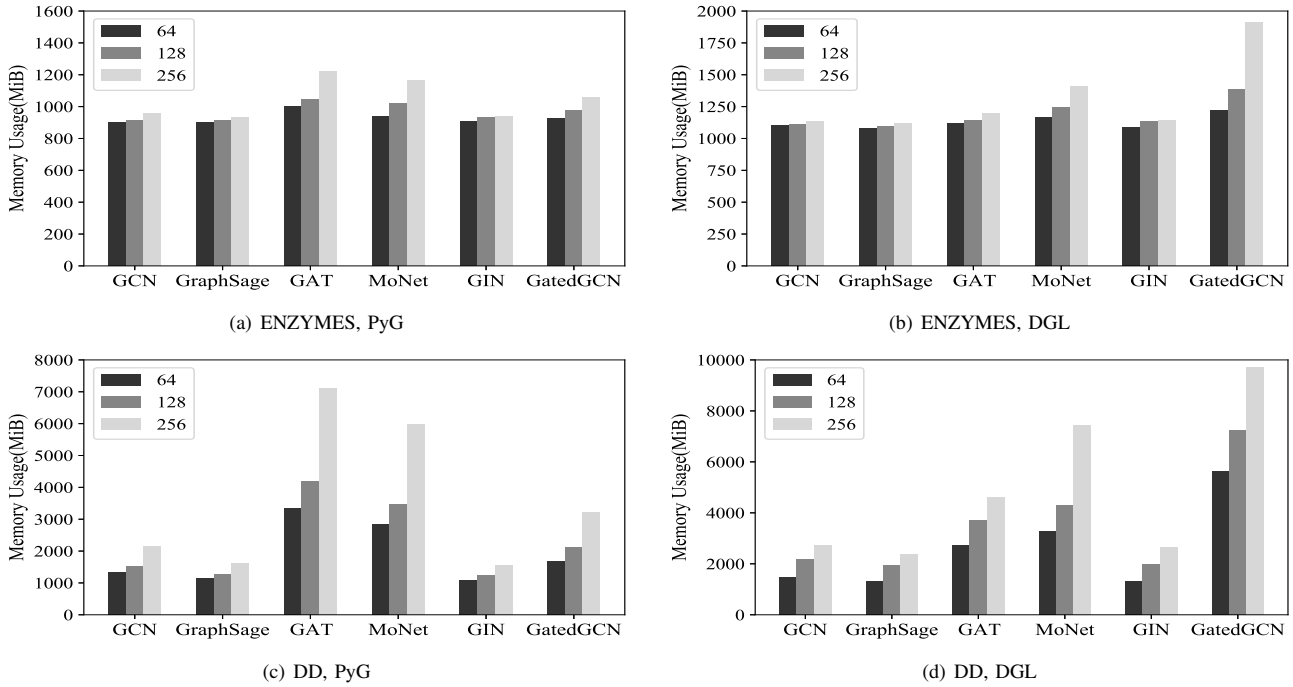
1) In general, because anisotropic GNNs employ complex mechanisms, they need more memory, which is more obvious on datasets with more nodes and edges. For anisotropic GNNs, when the batch size increases, the memory usage is improved more significantly.

2) In most cases, the memory usage of models implemented in DGL framework is more than PyG framework, but the gap is very big except for GatedGCN. In DGL, GatedGCN uses a fully connected layer to update features of all edges which needs a large amount of memory. And GatedGCN of PyG does not have the same operation.

3) The memory usage of most models implemented in the two frameworks is much lower than the provided memory and most models do not utilize available memory resources well. But for anisotropic models such as GAT and GatedGCN, the memory usage is quite large on datasets with a large number of nodes.

4) The GPU utilization of six models implemented in the two frameworks is low. For many cases, the maximum is no more than 40%. In general, these models do not utilize available GPU hardware resources well.

5) The GPU utilization of the six models implemented in DGL is slightly less than that of PyG for most cases. Among the six models in DGL, the memory usage and GPU utilization of GatedGCN is the biggest. In PyG, among the models, GAT has the biggest memory usage and GIN has the highest GPU utilization.



(a) ENZYMES, PyG

(b) ENZYMES, DGL

(c) DD, PyG

(d) DD, DGL

Fig. 4. Memory usage of different models with different batch size for two frameworks on dataset ENZYMES and DD.

6) Memory usage and GPU utilization is not the bottleneck of these models training on ENZYMES and DD, and further research should be done to optimize GNNs training on GPUs.

E. Multi-GPU Training

In this section, we characterize the performance scalability of GNN training using multiple GPUs. We use GCN and GAT models to do graph classification tasks on dataset MNIST in the two frameworks respectively. The two models belong to the anisotropic model and isotropic model respectively and can roughly present the training performance of these two types of models on multiple GPUs.

Training GNNs on large datasets can be accelerated by using multiple GPUs. This is usually achieved by using data parallelism, where mini-batches are split between individual GPUs and the results are then merged. To exploit data parallelism, a GPU gathers intermediate gradient values to update network parameters stored in other GPUs.

In our evaluations, the implementation of data parallelism of models is base on DataParallel API provided by PyTorch in the two GNN frameworks. We evaluate the time cost per epoch for GCN and GAT on dataset MNIST with multiple batch sizes on one, two, four, and eight GPUs. The results are presented in Fig.6.

The data loading of training GNNs on MNIST is pretty time-consuming and can take up more time than computing. Training models on multiple GPUs can only reduce the computing time and the computing time even can be reduced to 1/N when N GPUs are used. But training models on multiple GPUs can lead to the overhead of data transfer and when there are too many GPUs, the model training time will increase.

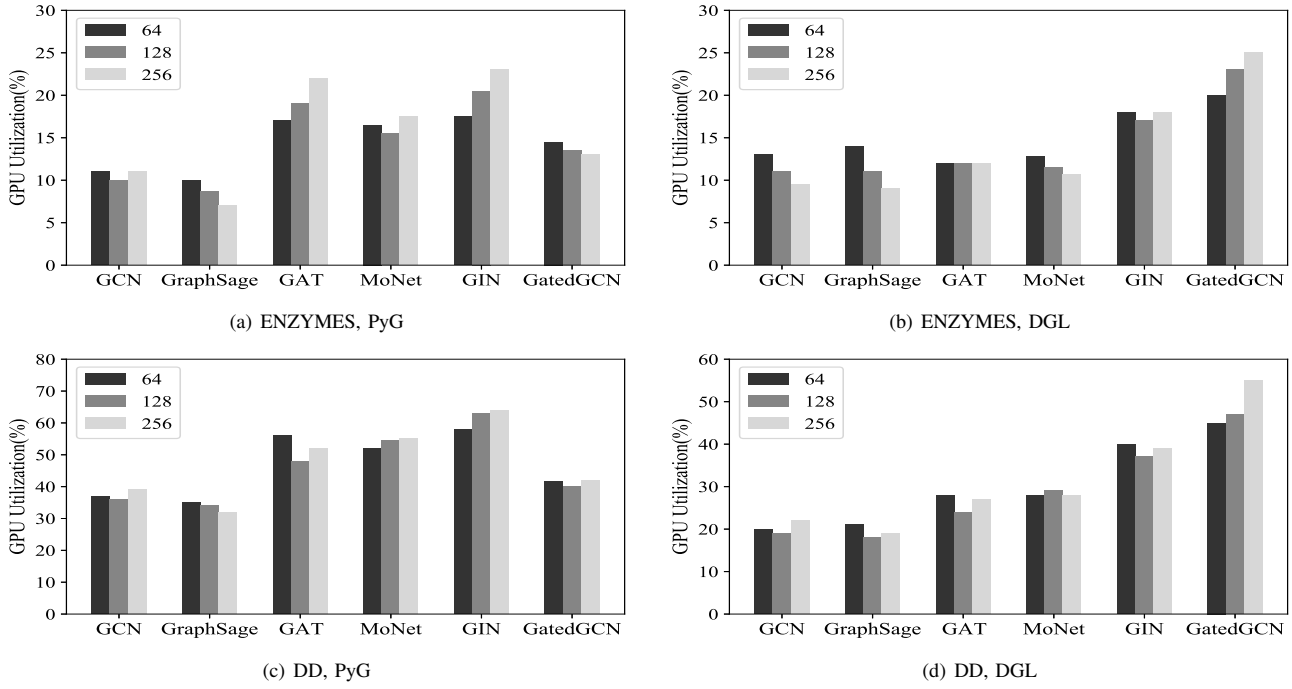
As Fig.6 shows, when the number of GPUs varies from 1 to 2 and from 2 to 4, the time cost per epoch slightly decrease with all the three batch sizes for two models across the two frameworks. This is because data loading time takes up the main part of training time and the nearly halved computing time can not bring about a significant reduction in the training time. And the reduction of computing time can make up for the overhead of data transfer. when the number of GPUs varies from 4 to 8, the time cost per epoch slightly increases in some cases and generally has no obvious reduction for most cases which is caused by the overhead of data transfer between the GPUs.

V. Conclusion

In this work, we evaluate the performance characteristics of six GNN workloads (GCN, GIN, GAT, GraphSAGE, MoNet, and GatedGCN) under two GNN frameworks (PyTorch Geometric and Deep Graph Library).

We perform a comprehensive evaluation and analysis to understand the training time performance, GPU utilization, and memory usage of models in the frameworks. We evaluate GPU utilization, and memory usage of workloads with different batch sizes under the two GNN frameworks. Besides, we present detailed execution times of models and evaluate the influences of hardware configurations (single-GPU and multi-GPU) on the GNN models under the two frameworks.

Based on our evaluation, models in the two frameworks have similar statistical test accuracy but the training time



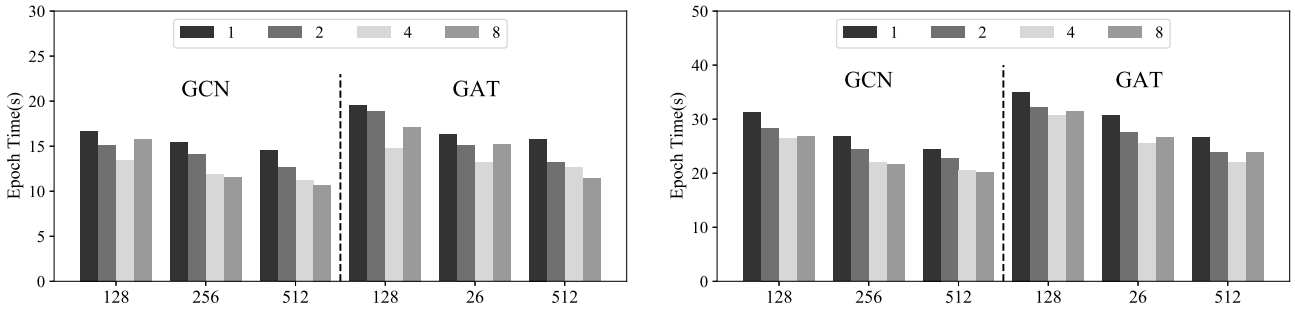
(a) ENZYMES, PyG

(b) ENZYMES, DGL

(c) DD, PyG

(d) DD, DGL

Fig. 5. GPU utilization of different models with different batch size for two frameworks on dataset ENZYMES and DD.



(a) PyG

(b) DGL

Fig. 6. Time cost per epoch with different batch size of multi-GPU training of the GCN and GAT models.

performance of models in DGL is significantly worse than that of PyG. In general, the memory usage of models in DGL is larger than that in PyG. For anisotropic models such as GAT and GatedGCN, memory usage is quite large on datasets with a large number of nodes and edges which may need new careful processing for memory usage.

The data loading time is a major part of training time because batching multiple graphs into a single large graph is pretty time-consuming, which also leads to the low GPU utilization of GNNs in the two frameworks. More efficient graph batching strategies will greatly speed up GNN training. On datasets with a large number of nodes and long node features, GNN models do not make good use of the parallelism of GPU training. The inter-node and intra-node parallelism of GNNs deserves further study.

REFERENCES

[1] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 1024–1034.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[3] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. A. Riedmiller, R. Hadsell, and P. W. Battaglia, "Graph networks as learnable physics engines for inference and control," vol. 80, pp. 4470–4479, 10–15 Jul 2018. [Online]. Available: http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html

[4] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and k. kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4502–4510.

[5] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 6530–6539.

[6] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, "Knowledge transfer for out-of-knowledge-base entities : A graph neural network approach," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 2017, pp. 1802–1808.

[7] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 6348–6358.

[8] D. Bacciu, F. Errica, and A. Micheli, "Contextual graph markov model: A deep and generative approach to graph processing," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 304–313.

[9] C. Wang, S. Pan, G. Long, X. Zhu, and J. Jiang, "MGAE: marginalized graph autoencoder for graph clustering," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*. ACM, 2017, pp. 889–898.

[10] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Trans. Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[11] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, vol. abs/1812.08434, 2018.

[12] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *CoRR*, vol. abs/1901.00596, 2019.

[13] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *CoRR*, vol. abs/1903.02428, 2019.

[14] y. Wang, Minjie and Yu, Lingfan and Zheng, Da and Gan, Quan and Gai, Yu and Ye, Zihao and Li, Mufei and Zhou, Jinjing and Huang, Qi and Ma, Chao and Huang, Ziyue and Guo, Qipeng and Zhang, Hao and Lin, Haibin and Zhao, Junbo and Li, Jinyang and Smola, Alexander J and Zhang, Zheng, journal=ICLR Workshop on Representation Learning on Graphs and Manifolds, "Deep graph library: Towards efficient and scalable deep learning on graphs." [Online]. Available: https://arxiv.org/abs/1909.01315

[15] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[16] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *CoRR*, vol. abs/1710.10903, 2017.

[17] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 1024–1034.

[18] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 5425–5434.

[19] X. Bresson and T. Laurent, "Residual gated graph convnets," *CoRR*, vol. abs/1711.07553, 2017.

[20] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[21] S. Ivanov, S. Sviridov, and E. Burnaev, "Understanding isomorphism bias in graph data sets," *CoRR*, vol. abs/1910.12091, 2019. [Online]. Available: http://arxiv.org/abs/1910.12091

[22] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, vol. 37, 2015, pp. 448–456.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, 2019, pp. 8024–8035.

[25] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[27] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014.

[28] NVidia, "cublas," http://docs.nvidia.com/cuda/cublas/index.html.

[29] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *Relational Representation Learning Workshop, NeurIPS 2018*, 2018.

[30] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[31] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *CoRR*, vol. abs/2003.00982, 2020.

[32] Hush and Salas, "Improving the learning rate of back-propagation with the gradient reuse algorithm," in *IEEE 1988 International Conference on Neural Networks*, 1988, pp. 441–447 vol.1.

[33] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. F. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, Ç. Gülçehre, H. F. Song, A. J. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. R. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018.

[34] Z. Zhu, S. Xu, J. Tang, and M. Qu, "Graphvite: A high-performance CPU-GPU hybrid system for node embedding," in *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, 2019, pp. 2494–2504.

[35] J. Jiang, L. Yu, J. Jiang, Y. Liu, and B. Cui, "Angel: a new large-scale machine learning system," *National Science Review*, vol. v.5, no. 2, pp. 102–122, 2018.

[36] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: a comprehensive graph neural network platform," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2094–2105, 2019.

[37] NVidia, "Profiler user's guide," 2018.

[38] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012.

[39] B. Weisfeiler and A. Leman, "A reduction of a graph to a canonical form and an algebra arising during this reduction (in russian)," *Nauchno-Technicheskaya Informatsia*, vol. 9, 01 1968.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.