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/ISPASS51385.2021.00013

# GNNMark: A Benchmark Suite to Characterize Graph Neural Network Training on GPUs

Trinayan Baruah[1] Kaustubh Shivdikar[1] Shi Dong[2] Yifan Sun[3] Saiful A Mojumder[4] Kihoon Jung[5]
José L. Abellán[6] Yash Ukidave[7] Ajay Joshi[4] John Kim[5] David Kaeli[1]
[1]Northeastern University [2]Cerebras Systems [3]William & Mary,[4]Boston University
[5]KAIST [6]Universidad Católica San Antonio de Murcia, [7]AMD
[1]{tbaruah,kaustubhcs,kaeli}@ece.neu.edu, [2]shi.dong@cerebras.net [3]ysun25@wm.edu, [4]{msam, joshi}@bu.edu
[5]jjk12@kaist.edu [5]jkih0021@gmail.com [6]jlabellan@ucam.edu [7]yash.ukidave@amd.com

***Abstract*—Graph Neural Networks (GNNs) have emerged as a promising class of Machine Learning algorithms to train on non-euclidean data. GNNs are widely used in recommender systems, drug discovery, text understanding, and traffic forecasting. Due to the energy efficiency and high-performance capabilities of GPUs, GPUs are a natural choice for accelerating the training of GNNs. Thus, we want to better understand the architectural and system-level implications of training GNNs on GPUs. Presently, there is no benchmark suite available designed to study GNN training workloads.**

**In this work, we address this need by presenting GNNMark, a feature-rich benchmark suite that covers the diversity present in GNN training workloads, datasets, and GNN frameworks. Our benchmark suite consists of GNN workloads that utilize a variety of different graph-based data structures, including homogeneous graphs, dynamic graphs, and heterogeneous graphs commonly used in a number of application domains that we mentioned above. We use this benchmark suite to explore and characterize GNN training behavior on GPUs. We study a variety of aspects of GNN execution, including both compute and memory behavior, highlighting major bottlenecks observed during GNN training. At the system level, we study various aspects, including the scalability of training GNNs across a multi-GPU system, as well as the sparsity of data, encountered during training. The insights derived from our work can be leveraged by both hardware and software developers to improve both the hardware and software performance of GNN training on GPUs.**

***Keywords*-graphs; benchmarks; GNN training; GPUs;**

## I. INTRODUCTION

Today, deep neural networks (DNNs) impact many fields, including image classification [1], speech recognition [2], and autonomous agents [3]. Popular DNN models, such as Convolutional Neural Networks (CNNs) [4] and Transformers [5], commonly operate on euclidean data. Euclidean data, as the name suggests, is data that lies in a flat 1D or 2D space. Image and speech data are two common examples of euclidean data. However, the vast majority of the data we interact with on a daily basis is non-euclidean [6]. Some of the common formats include molecules, social-network graphs, sensor-networks, and manifolds. Typical DNNs, which have been developed with euclidean data processing in mind, cannot handle non-euclidean data efficiently. This is due to the lack of efficient methods to apply operations directly (e.g., convolutions) to non-euclidean data [6].

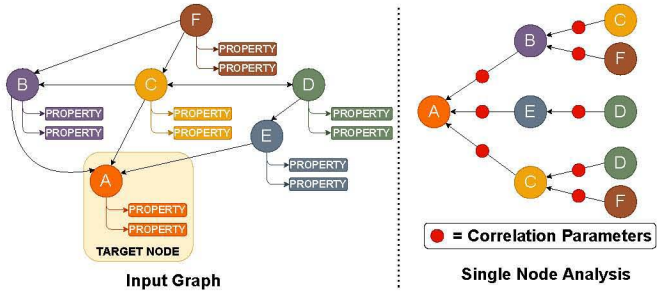


Fig. 1. Graph Neural Network Analysis

Graph Neural Networks (GNNs) [7], [8] have emerged to address this gap of training neural networks on non-euclidean data. The Pinterest social networking platform uses a GNN model named PinSAGE [9] to build its recommendation system. Researchers at Twitter are training GNN models [10] with temporal graph data. The Drug Repurposing Knowledge Graph (DRKG) [11] also employs GNN models to study the impact of existing drugs targeting new diseases.

Thanks to the advances enabled by using GPUs for DNN training, GPUs have become the de-facto platform for training GNNs as well. Currently, a majority of the popular GNN frameworks, such as the Deep Graph Library [12] and PyTorch Geometric [13], support GPU-based training in their backends. The growing popularity of GNN models demand efficient methods and platforms to train GNNs on GPUs. To develop efficient solutions for GNNs, it is critical to characterize the GPU's GNN training behavior in detail. Understanding the different types of GNN operations and further breaking down the training execution process can guide GPU developers to focus on bottlenecks and make informed design decisions. Additional characteristics, including the scalability of GNN training on multi-GPU systems and the degree of data sparsity during training, can be leveraged to train on larger graphs, especially those that cannot fit in a single GPU's memory [14]. Broadly speaking, characterizing GNN training workloads would help us develop a deeper understanding of the compute and memory characteristics of the GNN operations.

Prior work on characterizing GNNs has focused primarily on the inference behavior for Graph Convolutional Networks

978-1-7281-8643-6/21/$31.00 ©2021 IEEE
DOI 10.1109/ISPASS51385.2021.00013

(GCNs) [15] or targeted a limited set of GNN models [16]. Both model and dataset diversity [16] have not been considered by these studies. By dataset diversity, we mean different types of graphs, including homogeneous, heterogeneous, knowledge, and dynamic graphs (explored in detail in Section II). Model diversity implies different types of GNN models, such as Graph Transformers, Spatio-Temporal GNNs, and LSTM based GNNs. In previous benchmarking efforts, GNN inference has been the primary target for characterization studies [15], [16]. Popular benchmark suites for DNN training, such as the MLPerf Training Suite [17], Training Benchmarks for DNN (TBD) [18], DNNMark [19], Fathom [20], and DawnBench [21], do not consider GNNs as part of their workloads and deal exclusively with DNNs that deal with euclidean data. To comprehensively characterize the execution behavior of GNN training on GPUs, we need a benchmark suite that includes diverse GNN models that are trained on diverse datasets. Currently, no such benchmark suite exists. To fill this gap, we develop GNNMark, a collection of representative workloads that can be used by the computer architecture community to study the execution of GNNs on GPUs. We then analyze the workloads in the GNNMark benchmark suite, specifically focusing on their behavior during GNN training on a GPU. Apart from the fact that prior GNN workload characterization studies primarily focused on inference, we chose to focus on training given that GPUs remain the best platform in terms of performance for GNN training. In contrast, customized accelerators have been shown to outperform GPUs when executing GNN inference [22]

The contributions of our work include:

1. **We present the first GNN training-focused benchmark suite:** We deliver an open-source benchmark suite named GNNMark (https://gitlab.com/GNNMark/gnnmark), designed to characterize the training behavior of GNNs on GPUs. Our suite includes a diverse set of popular GNN models that have been developed by the machine learning community. **The workloads span seven different application domains and three different types of graph-based data types.**

2. **We carry out architecture-level characterization of GNNMark workloads during the training process:** We characterize the workloads in GNNMark, considering their architectural implications during the training process on a GPU. We are the first to provide a detailed execution time breakdown of different operations executed during GNN training and identify the major bottlenecks. We find that these workloads are much more diverse than typical DNN training workloads. GNN execution is highly input data and model dependent. We find that integer operations play a critical role, a factor which has been relatively ignored in DNN training studies on GPUs. We also observe significant sparsity during GNN training. This can potentially be leveraged to train larger graphs on a single GPU. We also consider multi-GPU support in the suite, enabling scaling studies of GNNs across multi-GPU systems.

3.**We provide recommendations to improve GPU architecture and overall system design:** We present insights drawn from our detailed characterization, and suggest changes to improve the GPU architecture and system design so that GNNs can be trained more efficiently on them.

## II. Background

### A. Graph Neural Networks

A Graph Neural Network (GNN) is a Machine Learning (ML) model designed to work on non-euclidean data, originally proposed to solve node classification problems [23]. The core idea behind using GNNs is to collect and aggregate information about a graph's structure to capture its inherent features and predict properties for specific nodes, connections, and generalizations to unseen graphs. Figure 1 (left) shows an example input graph with several edges connecting six nodes (A–F), where each node is represented by a set of feature vectors (properties).

Before a GNN model can make predictions, first it must be trained. As shown in Figure 1 (right), the goal of GNN training is to learn correlation parameters for each node, capturing its relation to the rest of the entire graph. More specifically, a feature vector for node A relates node A's properties to its neighbors' properties, nodes B, C and E. Each of these neighbors, in turn, have their own feature vectors to relate to their own neighbors. Hence the properties of node A can be associated with the properties of every other node in the graph. This feature of GNNs is also useful for finding missing properties of nodes in a graph. Similar to DNNs, a GNN can also have multiple layers, with each layer represented by two functions: i) an aggregation function and ii) an update function (i.e., a combination function). As the name suggests, the aggregation function is responsible for collecting or pooling the features of the neighbors for a given node. On the other hand, the update function is responsible for updating each node's feature vectors using Multi-Layer Perceptrons (MLPs). A GNN model can have layers with different aggregation and update functions. The deeper the GNN model, the more information a node has about other nodes that distant from it in the graph. However, training deeper GNNs is difficult, primarily due to the vanishing gradient problem [24]. As GNNs grow deeper, the gradients become so small that the weights stop getting updated. This property makes it difficult to train the GNN further. To address these challenges, novel GNN architectures have been proposed to enable deeper GNN models [25].

### B. Types of Graphs

GNNs have been explored in many different fields. Each type of GNN has inherently different types of graph data [8]. In our survey of GNNs, we observe that there are three such major categories of graph data:

**1. Homogeneous Graphs:** A homogeneous graph contains nodes and edges of a single type. For example, social network graphs are typically homogeneous, where each node represents a user, and an edge can represent that one user follows another. Homogeneous graphs can be directed (e.g., following a user on Twitter) or undirected (e.g., adding a friend on Facebook).

Another notable collection of homogeneous graph datasets that are used to evaluate GNN models are citation datasets (e.g., Cora, PubMed, Citeseer) [7].

**2. Heterogeneous Graphs:** A heterogeneous graph contains nodes and edges of multiple types. A widely used form of a heterogeneous graph is found in recommendation generation scenarios. For example, in a dataset designed to recommend music to users, the graph will consist of two types of nodes: i) music nodes and ii) user nodes. The edges will correspond to different interactions between the user and a music piece. In addition, edges may contain additional information such as ratings or like/dislike attributes. Knowledge graphs that are used to model relations between an object and entities are another form of a heterogeneous graph – e.g., when users search for a popular celebrity on Google (an object).

**3. Dynamic Graphs:** A dynamic graph is a special type of graph where the graph itself, as well as its properties, can evolve over time. Many real-world graphs, such as social-network graphs [10], traffic data graph [26] and communication-network graphs, are dynamic [27]. Note that dynamic graphs can be either homogeneous or heterogeneous. For example if we take a homogeneous social network graph, where nodes represent people and edges represent whether there is a relationship, the number of relations a person has or the relations between two people can change over time. Another common use case of dynamic graphs is to model traffic data as a dynamic graph and use it for traffic forecasting and prediction [28].

### C. Graph Neural Network Frameworks

Support for GNN primitives in popular ML frameworks is increasing. Today, researchers from the ML community are developing libraries in the form of extensions to frameworks such as PyTorch and TensorFlow. The two most popular libraries/extensions that implement customized GNN kernels, as well as provide programming support in the form of APIs, are PyTorch Geometric (PyG) [13] and the Deep Graph Library (DGL) [29]. PyG is an extension based on top of the PyTorch library and so only supports PyTorch. On the other hand, DGL provides support for PyTorch, TensorFlow, and MXNet. Spektral [30] and Aligraph [31] are two sets of libraries built on top of TensorFlow for GNN training. GraphNets [32] is a GNN framework from Google, supported using TensorFlow as the backend. As PyG and DGL bridge both the semantic and performance gaps when developing GNN models, they are the most widely used frameworks by both the ML community [33] and architecture community. [15], [16], [22], [34].

## III. Benchmark Suite Design

Characterizing the behavior of GNN training on a GPU requires a set of representative workloads to cover the wide variety of GNNs [8], [33]. The variants should include GNNs used across multiple application domains, including recommendation systems, classification of molecules, traffic forecasting, etc. The representative suite should also include models that consider different classes of real-world graphs, including knowledge graphs, heterogeneous graphs, and dynamic graphs. Also multi-GPU GNN training should be supported to evaluate the efficacy of training GNNs on multi-GPU systems.

To satisfy all the above-mentioned criteria, we present GNNMark, a benchmark suite designed for studying the behavior of GNN training on GPUs. Similar to benchmark suites that target DNN training, such as TBD [18] and MLPerf [17], we curate our benchmark suite from open-source publicly available implementations of GNN models. As PyTorch Geometric (PyG) and Deep Graph Library (DGL) are the main frameworks employed for developing GNN models by the ML community, we use models developed using these frameworks. Since both of these frameworks support PyTorch, we have chosen models developed in PyTorch. The specific models chosen for this suite, along with their associated application domains and datasets, are summarized in Table I. Below we provide more details about each GNN model.

**PinSAGE:** GNNs that operate on heterogeneous knowledge graphs can be used for recommendation tasks. These are commonly used in social networks. PinSAGE [9] is one such GNN model that has been developed at Pinterest. Since the original PinSAGE model is not publicly available, we use the implementation that has been published by the developers of DGL. PinSAGE is an improvement upon the GraphSAGE model [42] for training on large graphs. It uses a *random walk* mechanism [43] during aggregation to identify the importance of a node in the graph, without the need to process the entire graph. This effectively allows a user to train a model on graphs that do not fit in GPU memory.

**Spatio-Temporal Graph Convolutional Network:** Traffic forecasting is an important problem that falls into the domain of time-series prediction and uses dynamic graphs. This task is highly relevant for use in urban areas, where traffic control and guidance are required. Solving this problem using conventional euclidean-based DNNs is challenging because of the nonlinearity involved in traffic data [44]. One approach to deal with nonlinearity is to represent the problem as a graph and then apply depth-wise convolutions on the graph. Spatio-Temporal Graph Convolutional Networks (STGCN) [26] represent one such model that has been proposed to solve the problem of traffic forecasting. We include an STGCN to represent a GNN model that deals with dynamic graphs.

**DeepGCNs:** One of the key challenges with the original GCN models, such as the one proposed by Kipf and Welling [45], is that increasing the depth of the model does not improve the accuracy of the model. This is due to the vanishing gradient problem [24], which has made implementing deep GCNs challenging. Therefore, researchers have developed mechanisms to train deeper GCNs [25], using ideas borrowed from DNN research, such as residual layers and skip-connections used in models such as ResNet [46]. DeepGCN is a novel GCN architecture that allows GCNs to have more layers. Additional layers in a GCN can significantly improve training accuracy [25], so we include it in our study.

| Abbv | GNN Model | Application Domain | Graph Input Type | Datasets | # Node | # Edge |
|---|---|---|---|---|---|---|
| **PSAGE** | PinSAGE | Recommendation | Heterogeneous Graph | Nowplaying (NWP) [35] | 22.9M | 1.9M |
| | | | | Movielens (MVL) [36] | 1.9M | 9.7K |
| **STGCN** | Spatio Temporal GCN | Traffic Forecasting | Dynamic Graph | LA [26] | 207 | 325 |
| | | | | PEMS_Bay (PEMS) [26] | 1722 | 2694 |
| **DGCN** | Deep GCN | Molecular Property Prediction | Homogeneous Graph | MOLHIV [37] | 1.04M | 1.1M |
| | | | | MOLTOX [37] | 145K | 151K |
| **GW** | GraphWriter | Text Generation | Heterogeneous Graph | AGENDA [38] | 885K | 2.57M |
| **KGNN** | k Graph Neural Networks | Protein Classification | Homogeneous Graph | Proteins (PROT) [39] | 43K | 162K |
| **ARGA** | Adverserially Regularized Graph Autoencoder | Node Clustering | Homogeneous Graph | Cora [40] | 2K | 10.5K |
| | | | | CiteSeer (CSEER) [40] | 3.3K | 9.2K |
| | | | | PubMed (CSEER) [40] | 19.7K | 88.6K |
| **TLSTM** | Tree Long Short-Term Memory Networks | Sentiment Classification | Homogeneous Graph | Stanford Sentiment Treebank (SNTM) [41] | 318K | 310K |

TABLE I
WORKLOADS IN GNNMARK BENCHMARK SUITE.

Specifically, we use a DeepGCN model and train it to perform graph property prediction, a common task in molecular property prediction.

**GraphWriter:** Automated generation of text from a knowledge graph to form meaningful and coherent sentences is an open and challenging problem [47]. Text encoding models, such as the popular Transformer model [5], cannot be directly applied to a knowledge graph as they do not work with non-euclidean data. Therefore, ML researchers have developed GNN-based Transformer models for this task. Graphwriter [38] is one such novel GNN-based Transformer model, designed to operate on knowledge-graphs for text generation.

**k-GNNs:** Most GNN models are one-dimensional in nature and cannot effectively capture any higher-order information, such as the properties of subgraphs, within the graph. As a result, they fail the graph isomorphism test proposed by Weisfeiler and Lehman [48] (WL algorithm). The WL algorithm is a test used to determine the expressiveness power of a GNN by testing if an algorithm is able to distinguish whether two graphs are isomorphic or not. Two graphs are said to be isomorphic if they have the same number of vertices, edges, and connectivity. Therefore, researchers have developed higher-dimensional hierarchical GNNs, called k-GNNs (where the k stands for the dimension), which can capture properties of subgraphs [49]. This enables GNNs to perform close to the k-WL graph isomorphism test [49]. We include two variants of k-GNNs, (KGNNL and KGNNH to denote a lower and higher dimensional version of k-GNN, respectively) and use them to perform classification of protein molecules. The primary reason we include this workload in our suite is to study how application characteristics and behavior change as we move towards higher-dimension GNNs.

**Adversarially Regularized Graph Autoencoder:** Generative Adversarial Networks (GANs) are gaining popularity due to their ability to learn with limited amounts of data [50]. Due to this property, GAN-based architectures are also being explored for GNNs.Adversarially Regularized Graph Autoencode (ARGA) [51] is one such GNN-based GAN model that is proposed for graph embedding. ARGA has an encoder-decoder architecture where the encoder is trained to form a compact representation of a graph, and the decoder is trained to generate the graph structure. The model is designed to perform node clustering, which is an unsupervised learning task, on real-world graphs. ARGA employs this encoder-decoder architecture within a GAN framework, so that it can successfully learn the low-dimensional features of the graph from the high-dimensional graph features. This process is referred to as graph embedding [52]. We include ARGA as a representative GAN-based GCN to further increase the diversity of our benchmark suite. We train ARGA to perform node clustering on real-world homogeneous graphs, such as Cora, PubMed, and CiteSeer [45].

**Tree-LSTM:** Sentiment classification is an important task in the Natural Language Processing (NLP) domain. Tree Long Short-Term Memory Networks (Tree-LSTMs) [53] are one group of models developed for this task. In contrast to the linear model used in an LSTM, Tree-LSTMs use a tree-structured network topology and can outperform linear LSTMs in the sentiment classification task [53]. The Tree-LSTM method implemented in DGL uses the idea of batching. The basic idea of batching is to collect smaller graphs that are part of the dataset and convert them into a batched larger graph. We include the Tree-LSTM model in GNNMark to study how batching of multiple small graphs to a larger graph impacts the behavior of an application.

## IV. METHODOLOGY

### A. Experimental Platform

To demonstrate the utility of GNNMark, we use an NVIDIA V100 [54], a commonly used GPU for running neural network training. V100 is part of the NVIDIA Volta family of GPUs. Our test system is equipped with an Intel(R) Xeon(R) CPU

E5-2630 CPU that operates at a frequency of 2.4GHz. The GPU has 80 Streaming Multiprocessors (SMs) and is rated to deliver 14 TFLOPS of single-precision performance. The GPU memory uses HBM2 with 16 GB capacity and bandwidth of 900 GB/s. The combined L1 cache/shared memory/texture cache has a capacity of 128 KB and is private to each Streaming Multiprocessor (SM). The L1 memory is backed by a 6.14 MB L2 cache, which is banked and shared across all SMs.

For our multi-GPU experiments, we use 4 V100 GPUs on a node equipped with Intel(R) Xeon(R) E5-2686 v4 2.4GHz CPUs, hosted on Amazon AWS EC2. Each GPU is interconnected using NVIDIA NVLink technology, providing a total of six links, for an aggregate bandwidth of 300 GB/s. Both the single-GPU and multi-GPU systems used in our experiments run CUDA 10.2, cuDNN 7.6.5, and PyTorch 1.5.0. The workloads included in GNNMark use either DGL version 0.5.2 or PyTorch Geometric 1.6.1, which are the latest versions of these libraries at the time of writing this paper.

Since multi-GPU training has been shown to improve the performance of DNN training [55], we also look at how well GNN training can scale across multiple GPUs. GNN training can be sometimes be limited by GPU memory capacity, especially given the continual growth in size of the input graph [56]. One approach to counter this problem is to compress the data transferred from the CPU to the GPU and store the compressed data in GPU main memory. This is only possible if the data transferred is highly sparse [14]. Therefore, we also characterize sparsity levels of the data transferred between the CPU and GPU during GNN training in our suite.

### B. Profiling Tools

We use several tools for collecting the metrics of interest. For the kernel-level characteristics, such as cache statistics and comparisons between compute versus memory behavior, we use the NVIDIA nvprof profiler (version 10.2) [57]. Similar to DNNs, GNNs typically launch the same kernel many times during training. Therefore, when profiling and collecting hardware performance counters using nvprof, we profile the same kernel for either fifty kernel invocations or for one epoch, whichever is shorter. However, nvprof does not provide any mechanism to collect memory divergence behavior of a workload. Therefore, we use the NVBit framework [58] (version 1.4), which is a binary instrumentation tool to collect the memory divergence information at a kernel level. To collect the sparsity of the data transferred from the host to the device during GNN training, we modified the PyTorch source code to collect this information.

### C. Multi-GPU Implementations

We also include multi-GPU versions of each workload in GNNMark to enable users to study the scalability of GNN training on multi-GPU systems, as well as understand how well high-level support for multi-GPU DNN training in frameworks such as PyTorch scale in practice for GNNs. The multi-GPU implementations are built on the PyTorch `DistributedDataParallel` (DDP) method to train GNNs across multiple GPUs, exploiting data-level parallelism. DDP has been shown to scale well, in practice, on up to 256 GPU nodes [59] for DNN training.

## V. Results

### A. Execution Time Breakdown

We start our analysis by breaking down the time spent in the different GNN operations across the different workloads in our suite. Similar to DNNs [20], GNN training can be broken down into layers or operations. Prior work divided GNN training into two phases: i) an aggregation phase, and ii) an update phase [15]. While this division is appropriate when looking at GNNs from an ML perspective, we believe that deeper insights are needed to fully characterize their behavior. Therefore, we work at the abstraction level of individual operations [20].

We identify a common set of operations performed during GNNMark execution. These operations include general matrix multiply (GEMM), sparse matrix matrix multiplication (SpMM), convolutions, scatters, gathers, reductions, index selection, sorting, and element-wise operations. Element-wise operations are operations that operate on individual elements of a tensor and perform operations such as multiplication of all elements in the tensor by a scalar, change the sign of all elements in the tensor, adding two tensors of similar dimensions etc

Figure 2 shows a breakdown of the percentage of time spent in individual operations across the different workloads of GNNMark. We can see from Figure 2 that the percentage breakdown of operations varies significantly across workloads. For instance, STGCN, a spatio-temporal GNN, is dominated by 2D convolution operations (60% on average), while DGCN is dominated by element-wise operations (31% on average).

The execution time breakdown across operations in a GNN differs greatly from the mix in a typical DNN. Across all the workloads, we observe that only 25% of the execution time is spent executing GEMM and SpMM operations. This is in stark contrast to the mix of operations common in DNN workloads, where GEMM (convolutional layers and fully-connected layers) dominate the execution [19]. We find that GNN training also differs significantly from GNN inference workloads [15], where GEMM operations are reported to consume more than 50% of the execution time.

Other common operations, such as sorting, index selection, reductions and scatter-gather operations, account for 20.8% of the total execution on average. These operations are primarily used in the graph's aggregation phase, where the nodes exchange information with one another before updating the feature vectors.

PSAGE, when trained on the MVL dataset, spends 20.7% of its execution on sorting, and only 7.0% of the time on reductions, whereas ARGA (with Cora data) spends 23% on reductions and only 6.1% on sorting. This great diversity and variety of tasks in GNN training present challenges to architects designing customized accelerators for GNN training

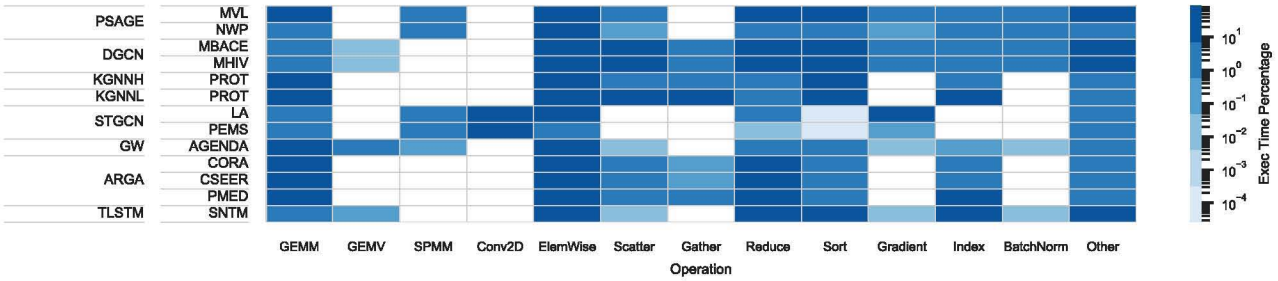


Fig. 2. Execution breakdown, reported as the percent of total execution time, for individual operations across the different workloads of GNNMark.

given that accelerators are typically designed to optimize only for a single set of operations.

In contrast to typical DNN workloads, GNN workloads tend to be more input data-dependent. For PSAGE, the percentage of time spent in element-wise operations is much higher when training on the (NWP) dataset (78%), versus training on the (MVL) dataset (36%). This is because, when training on the NWP dataset, the feature vectors are 10× larger than when training on the MVL dataset. As element-wise operations operate on each value of the input feature vector, the time spent executing these operations becomes more dominant when graphs with larger input features are used.

**Takeaways**: Our performance analysis shows that GNN training workloads exhibit more diverse behavior as compared to DNN training workloads. Each model's characteristics can differ vastly from others. Even the same GNN model can exhibit different characteristics depending upon the input graph type. In addition, execution hot spots are no longer limited to convolution and GEMM operations. We find operations such as reductions, scatter, gather and sorting also need to be optimized. The solution of attaching a single-purpose accelerator to primarily accelerate GEMM operations [60] during DNN training may not work well for GNN training.

### *B. Instruction Mix and GFLOPS/GIOPS Analysis*

Another aspect of GNN training behavior is the dynamic instruction mix present in different workloads. As we can see in Figure 3, integer instructions play a larger role than floating-point instructions across all workloads. On average, 64% of the executed instructions are integer (int32) instructions, whereas only 28.7% are single-precision floating point (fp32) instructions. The only workload where this trend is reversed is in GraphWriter (GW). This is because in GW, a majority of the time is spent on GEMM and SpMM operations (as seen in Figure 2) which work on fp32 data. While improving the performance of fp32 instructions has received a lot of attention, int32 instructions have not received the same. Given that int32 instructions dominate GNN execution during training, improving the performance of integer math on a GPU is a critical factor when trying to accelerate GNN training.

Figure 4 presents the number of GFLOPS and GIOPS executed by our workloads in GNNMark. We observe that the average GFLOPS rate is 214 GFLOPS, and the average GIOPS rate is 705 GIOPS. The observed average GFLOPS rate is much lower than the theoretical max GFLOPS of the V100, which is 14 TFLOPS for fp32 arithmetic [54] (the V100 specs do not mention the peak theoretical GIOPS. We believe it to be close to the peak theoretical GFLOPS). GW has the highest fp32 performance of 1.99 TFLOPS. Being a transformer-based ML model, GW can effectively use most of the parallel resources on a GPU [5]. We also observe that, while graph batching has been proposed to improve performance in DGL, TLSTM is still able to achieve only 74 GFLOPS.

The average IPC measured across all the workloads was found to be 0.55, which reflects the memory-bound nature of the workloads. When comparing the GFLOPS and GIOPS of different operations, we observe that the GEMM operations typically have a higher GFLOPS (in the mid 300's) as opposed to other operations, such as reductions, scatters and gathers that have lower rates (in the 100 GFLOPS/GIOPS range) suggesting a very low overall GPU utilization. Given that these operations can dominate the execution time, it is important for both hardware and software developers to focus on improving the performance of these operations.

**Takeaways**: Our analysis reveals that, during GNN training, execution is dominated by integer operations. Thus, to accelerate GNN training on either GPUs or accelerators, int32 arithmetic performance will be key. The overall performance in terms of GFLOPS/GIOPS for GNNs is relatively low compared to the peak performance of the hardware. This suggests that GNN training is primarily memory bound. Given that operations such as reductions, scatters, gathers and sorting can occupy a good chunk of the execution time during GNN training, it is important for both hardware and software developers to focus on improving the performance of these operations.

### *C. Stalls and Cache Analysis*

Developing a comprehensive understanding of major stalls in the GPU hardware during GNN training can help guide architectural design decisions when tuning the performance of these workloads. Given that caches can greatly improve the performance of GPU applications, it is also important to look at their efficiency in the context of GNN training.

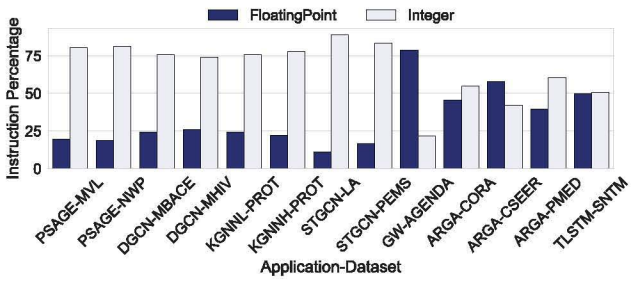


Fig. 3. Breakdown of fp32 vs. int32 instructions across the different workloads in GNNMark.

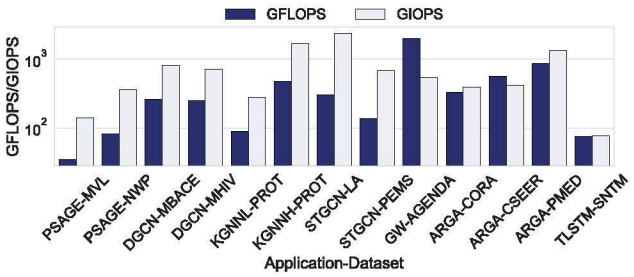


Fig. 4. GFLOPS and GIOPS across the different workloads in GNNMark.

In Figure 5 we show the distribution of different types of stalls observed in GNN training. We find that execution is stalled primarily due to *Memory Dependency*, *Execution Dependency* and *Instruction Fetch*. The high percentage of *Memory Dependency* stalls (34.3% on an average) suggests that the memory subsystem is inefficient in serving data read requests to the GPU cores. From Figure 6, we observe that GNN workloads have an extremely low L1 D-cache hit rate on the V100 (a mere 15%, on average) which is the primary reason for these stalls.

We also analyze the impact of divergent load instructions. The load instructions associated with a warp are said to be divergent if they access more than one cache line (a line is 128B on the V100). Memory divergence can impact the performance of typical graph workloads, such as Breadth First Search and PageRank [61]. Therefore, it is important to characterize the degree of memory divergence present during GNN training.

We observe 32.5% of divergent load instructions, on average, across the different workloads. This percentage is large and is highly correlated with resulting low L1 D-cache hit rates. While the larger L2-cache (6MB) on the V100 fares significantly better with a 70% hit rate on average, the inability of the L1 D-cache to effectively hold the working set can put pressure on the L2 cache to satisfy the memory requirements. Across the different operations, we observe that GEMM, SpMM and GEMV have poor locality (i.e., a low L1 D-cache hit rate, less than 10% on average). The L1 D-cache hit rates of other operations, such as indexing, scatters, gathers and sorting, are also low (below 15%, on average).

The high percentage of *Execution Dependency* stalls (i.e., 29.5% on an average) points to the fact that across the

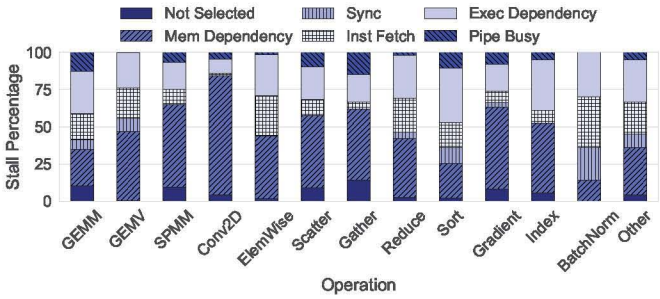


Fig. 5. Stall breakdown across operations in GNNMark.

entire set of workloads, there are many dependencies between instructions in a warp, which results in low instruction-level parallelism. Microarchitectural enhancements to support out-of-order execution in the GPU pipeline [62] can therefore potentially accelerate GNN training.

Surprisingly, *Instruction Fetch* stalls are also significant (21.6% on an average). This is due to two reasons. The first is that the instruction cache is ineffective in caching all the instructions. Although the V100 architecture has a new 12KB L0 I-cache that is backed by a larger 128KB L1 I-cache, it seems to not be highly effective in caching all instructions during kernel execution. The second reason is that loop unrolling techniques [63], which are used to improve the performance of a GPU kernel, can negatively impact the instruction cache hit rate and increase the stalls due to instruction fetching [64].

As observed in Figure 5, for commonly used GNN operations (Conv2D is used only for STGCN and BatchNorm for DeepGCN), the scatter and gather operations, index selection operations have a higher rate of stalls when compared to GEMM due to memory dependencies. This is primarily because both scatter and gather, as well as index selection operation, exhibit an irregular memory access pattern.

---

**Takeaways**: Our analysis of the stalls during GNN training shows that stalls due to *Memory Dependency*, *Execution Dependency* and *Instruction Fetch* can be significant. While GPU architecture research has focused on removing the first two types of stalls, improving the performance of instruction fetching has been neglected. Therefore, architects and compiler developers should focus on developing techniques to improve instruction fetching to optimize the performance of GNN training.

GNN training also suffers from a high degree of L1 D-cache misses and a significant number of divergent load instructions across all operations. The extremely high L1 D-cache miss rates suggest that caching is not effective for GNN workloads. We envision two potential solutions to alleviate this problem. The first is to employ half-precision-training for GNN training, which uses only 16-bit data instead of 32-bit data and can thereby reduce the L1 D-cache miss rates. Alternatively, L1 cache bypassing solutions [65], [66] can be explored to alleviate this problem.

---

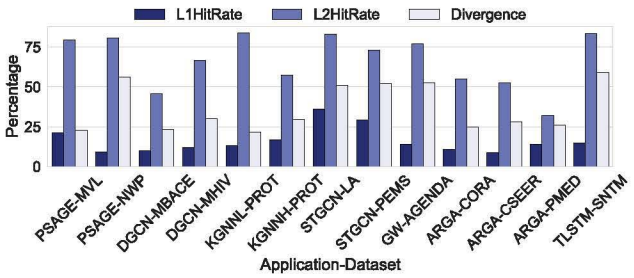


Fig. 6. L1-D and L2-cache hit ratios, and divergent load ratios for GNNMark workloads.

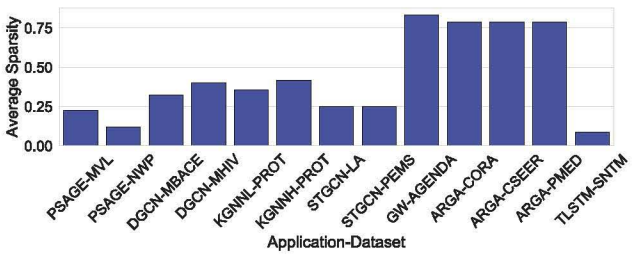


Fig. 7. Average sparsity in the data transferred from CPU-to-GPU during GNN training in GNNMark workloads.

### D. Sparsity during GNN training

Training sparsity refers to the zero values (as a percentage of all values) that are transferred during CPU-to-GPU memory copies during the GNN training process. For characterizing the average sparsity, we report the percentage of zero values observed in CPU-to-GPU data transfers during GNN training. From Figure 7, an average sparsity of 43.2% was observed during GNN training. This suggests that compression techniques could be employed. Rhu et al. [14] proposed using compression to alleviate the problem of training large DNN models on a GPU. While GNN models are smaller than conventional DNN models (e.g., Resnet-50 is 50-layers deep, whereas most GNNs today have fewer than 10 layers), the input graph can occupy a significant portion of GPU memory (up to 90% in our experiments). While the ML community has proposed sampling the graph to address this problem [9], there are situations where training on the whole graph has been shown to provide better accuracy [56]. We suggest compressing the data in GPU memory to facilitate training on large graphs.

We can also observe a clear, predictable pattern in the data sparsity (from Figure 8), providing opportunities to apply adaptive compression algorithms [67]. As the sparsity values change during training, the GNN training framework may need to exploit different compression solutions and formats that work the best for a specific sparsity level.

Looking at the average sparsity for `PSAGE` in Figure 7, we can conclude that training sparsity is a function of both the model and the input graph. When using the MVL dataset, the average sparsity is 22%, but it reduces to 11% when training on the NWP dataset.

In terms of models, since many GNNs such as Graph-Transformer, DeepGCNs and ARGA use activation functions such as ReLU and PReLU in their layers, they produce highly sparse data. We suggest applying compression to take advantage of this sparsity. The result will be that we can train larger graphs on a single GPU. We plan to pursue this path in future work.

**Takeaways**: Training on graphs that are larger than the size of GPU memory is a challenging problem. Thus, exploiting the high degree of sparsity present in GNN workloads by using compression techniques can begin to address this problem.

### E. Scalability of GNN training using multi-GPU systems

Using the multi-GPU implementations that we developed for the GNNMark workloads using PyTorch DDP, we evaluate the strong scaling characteristics of the workloads in GNNMark. We train all our models for five epochs (we observe similar performance across all epochs) and report the average `time-per-epoch`, an approach used in previous work [55], to understand the performance of DNN workloads on multi-GPU systems. We do not evaluate ARGA, as the application inherently sends the entire graph to the GPU as a part of its training process, and therefore distributing the same graph across multiple GPUs does not help. The first thing we can clearly observe from Figure 9 is that not all workloads benefit from multi-GPU training. While DGCN,STGCN and GW show considerable performance gains, the same does not hold true for the other applications. TLSTM does not benefit from multi-GPU training. Given that this is an LSTM-based GNN model with low computational GFLOPS/GIOPS intensity, the application is unable to take advantage of the additional computing power offered by multi-GPU systems. For PSAGE, we observe performance degradation when scaling across multiple GPUs. This is primarily because the PSAGE implementation in DGL uses a batch sampling mechanism which is not compatible with PyTorch DDP. As a result, the training data gets replicated across multiple devices and this replication results in redundant computation and unnecessary communication which in turn hurts performance.

**Takeaways**: Multi-GPU systems do not always benefit GNN training. Therefore, ideas such as topology-aware scheduling and fine-grained graph partitioning that have been proposed by researchers in graph-centric GNN frameworks, such as ROC [56] and NeuGraph [68], should be adopted by high-level frameworks, such as PyG and DGL, to enable more efficient GNN training. Currently, these frameworks are not open source, and hence, we cannot evaluate them for the GNNMark workloads.

## VI. Related Work

Past GPU benchmark suites have provided guidance to GPU architects. To date, GPU benchmarks fall into one of

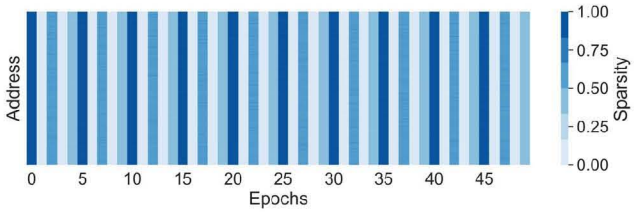


Fig. 8. Sparsity heat map for DeepGCN when running on the MOLHIV dataset.

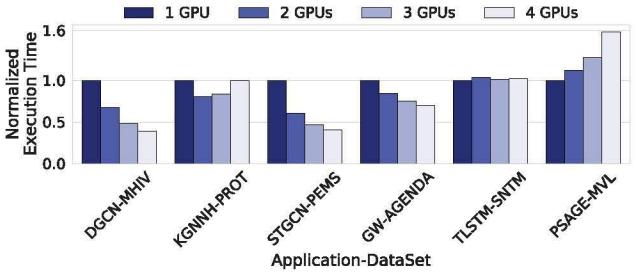


Fig. 9. Multi-GPU performance scaling.

two categories. They either evaluate general-purpose GPU computing capabilities [61], [69]–[73], or target assessment of the performance of a specific class of workloads [74]–[76]. With the growing popularity of DNN workloads, a new wave of DNN benchmarks have been developed.

**Benchmarking Deep Learning Workloads and Workload Characterization:** Early DNN benchmark suites explored the execution performance of low-level primitives [19], [77], as well as end-to-end inference and training [20], [21]. Later efforts included a more diverse set of DNN algorithms, including a broader range of network models and commercial efforts. TBD [18] is a DNN benchmark suite proposed by Zhu et al. to study DNN training performance on GPUs. AIBench [78] is an industry-initiated benchmark suite that is focused on industrial AI services. Mattson et al. [79] proposed the MLPerf training and MLPerf inference suites. MLPerf adopts ideas from prior DNN benchmark suites to develop an industry-standard DNN-focused benchmark suite, designed so that new hardware and software optimizations can be evaluated fairly. To date, MLPerf has primarily focused on DNNs. In terms of architectural characterization, Dong et al. [80] looked at the architectural implications of CNN training on a GPU. Mojumder et al. [55] profiled DNN models trained on an NVIDIA DGX-1 system. However, all these prior workload studies were limited to DNNs that operate on euclidean data (e.g., images, video and speech). GNNMark is designed specifically to bridge this gap, providing the architecture community with an appropriate benchmark suite to study GNN training behavior. We also plan to work with the MLPerf consortium to integrate our GNN models into their training suite in the future.

**Workload Characterization for GNNs:** GNNs have recently attracted attention from the computer architecture community due to their growing popularity in the machine learning domain. Yan et al. [15] have characterized Graph Convolutional Network (GCN) inference performance, focusing on aggregation and model update phases. Zhang et al. [16] have also characterized the inference performance of GNNs. Their work decomposes GNN inference execution into a Scatter-ApplyEdge-Gather-ApplyVertex (SAGA) pipeline and then analyzes the behavior of each phase. They also present insights on how to efficiently design a GNN accelerator for inference. While a benchmark suite is also created as a part of their study, it is designed primarily for inference and is not available publicly. Most prior GNN studies focused on inference, ignoring the training process that tends to consume a large number of GPU hours. Also, the models evaluated are only designed to process homogeneous graphs. Other related work focused on characterizing GNN inference and designing customized accelerators for that purpose [22], [34], [81]. In contrast, GNNMark includes GNN models that work across a wide range of graph data, including spatio-temporal graphs and heterogeneous graphs. GNNMark also includes multi-GPU implementations of GNNs, making it suitable for research on GNN training behavior targeting GPUs. The applications included in GNNMark can also be used to drive inference studies by first training the models to a target accuracy, and then using the pretrained models to characterize inference. We plan to extend the suite to support inference studies by providing a set of pretrained models in the future.

Dwivedi et al. [82] propose a benchmark suite targeting GNN models for the machine learning community. In contrast to GNNMark, which is designed to drive architectural studies, their benchmark suite is more focused on comparing the accuracy and performance of different popular GNN models in the literature, evaluating the models against a standard collection of graph datasets.

## VII. Conclusion and Future Work

In this paper we present GNNMark, a diverse benchmark suite of GNN workloads designed for characterization of GPU performance. To the best of our knowledge, we are the first to propose a GNN-training focused benchmark suite for the architecture community. We use GNNMark to perform a detailed characterization of GNNs to understand the architectural implications of training on GPU systems. Our work provides novel insights that show what are the major architectural bottlenecks in GNN training and suggests how they can be potentially addressed.

Clearly, a single GNN model can exhibit different characteristics, based on the input graph. We also observe that, unlike DNNs, GEMM and convolution operations are less dominant in GNN execution. Instead, integer operations required for graph processing can dominate execution, suggesting that improving the performance of integer math is paramount. A high degree of instruction fetch stalls show that the instruction cache on the GPU can limit GNN performance. Finally, we also report on the training sparsity and strong scaling characteristics of GNN training using our suite.

For future work, given the high sparsity found in these workloads, we plan to explore using compression to accelerate

GNN training on large graphs, as well as understand the weak scaling characteristics of GNN training. We also plan to update our suite using the `time-to-train` metric proposed by the developers of MLPerf [79] and support half-precision-training in GNNMark. We also plan to add more models, such as those in the realm of Reinforcement Learning based GNNs [83], to GNNMark.

## Acknowledgements

We thank the anonymous reviewers for their constructive feedback. We also thank AMD for supporting this work.

## References

[1] P. Druzhkov and V. Kustikova, "A survey of deep learning methods and software tools for image classification and object detection," *Pattern Recognition and Image Analysis*, vol. 26, no. 1, pp. 9–15, 2016.

[2] J. Padmanabhan and M. J. Johnson Premkumar, "Machine learning in automatic speech recognition: A survey," *IETE Technical Review*, vol. 32, no. 4, pp. 240–251, 2015.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

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

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

[8] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[9] 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*, 2018, pp. 974–983.

[10] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[11] V. N. Ioannidis, X. Song, S. Manchanda, M. Li, X. Pan, D. Zheng, X. Ning, X. Zeng, and G. Karypis, "Drkg-drug repurposing knowledge graph for covid-19," 2020.

[12] D. Zheng, M. Wang, Q. Gan, Z. Zhang, and G. Karypis, "Learning graph neural networks with deep graph library," in *Companion Proceedings of the Web Conference 2020*, 2020, pp. 305–306.

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

[14] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing dma engine: Leveraging activation sparsity for training deep neural networks," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 78–91.

[15] M. Yan, Z. Chen, L. Deng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Characterizing and understanding gcns on gpu," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 22–25, 2020.

[16] Z. Zhang, J. Leng, L. Ma, Y. Miao, C. Li, and M. Guo, "Architectural implications of graph neural networks," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 59–62, 2020.

[17] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf *et al.*, "Mlperf training benchmark," *arXiv preprint arXiv:1910.01500*, 2019.

[18] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Benchmarking and analyzing deep neural network training," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 88–100.

[19] S. Dong and D. Kaeli, "Dnnmark: A deep neural network benchmark suite for gpus," in *Proceedings of the General Purpose GPUs*, 2017, pp. 63–72.

[20] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, "Fathom: Reference workloads for modern deep learning methods," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1–10.

[21] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Dawnbench: An end-to-end deep learning benchmark and competition," *Training*, vol. 100, no. 101, p. 102, 2017.

[22] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 15–29.

[23] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[24] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[25] G. Li, M. Müller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[26] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

[27] D. Eppstein, Z. Galil, and G. F. Italiano, "Dynamic graph algorithms," *Algorithms and theory of computation handbook*, vol. 1, pp. 9–1, 1999.

[28] Z. Diao, X. Wang, D. Zhang, Y. Liu, K. Xie, and S. He, "Dynamic spatial-temporal graph convolutional neural networks for traffic forecasting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 890–897.

[29] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.

[30] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral," *arXiv preprint arXiv:2006.12138*, 2020.

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

[32] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[33] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[34] S. Liang, Y. Wang, C. Liu, L. He, L. Huawei, D. Xu, and X. Li, "Engn: A high-throughput and energy-efficient accelerator for large graph neural networks," *IEEE Transactions on Computers*, 2020.

[35] E. Zangerle, M. Pichl, W. Gassler, and G. Specht, "# nowplaying music dataset: Extracting listening behavior from twitter," in *Proceedings of the first international workshop on internet-scale multimedia management*, 2014, pp. 21–26.

[36] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[37] 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.

[38] R. Koncel-Kedziorski, D. Bekal, Y. Luan, M. Lapata, and H. Hajishirzi, "Text Generation from Knowledge Graphs with Graph Transformers," in *NAACL*, 2019.

[39] K. Kersting, N. M. Kriege, C. Morris, P. Mutzel, and M. Neumann, "Benchmark data sets for graph kernels," 2016. [Online]. Available: http://graphkernels.cs.tu-dortmund.de

[40] Z. Yang, W. Cohen, and R. Salakhudinov, "Revisiting semi-supervised learning with graph embeddings," in *International conference on machine learning*. PMLR, 2016, pp. 40–48.

[41] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, 2013, pp. 1631–1642.

[42] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[43] W. Wei, J. Erenrich, and B. Selman, "Towards efficient sampling: Exploiting random walk strategies," in *AAAI*, vol. 4, 2004, pp. 670–676.

[44] A. S. Nair, J.-C. Liu, L. Rilett, and S. Gupta, "Non-linear analysis of traffic flow," in *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No. 01TH8585)*. IEEE, 2001, pp. 681–685.

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

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[47] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu, "A survey on knowledge graphs: Representation, acquisition and applications," *arXiv preprint arXiv:2002.00388*, 2020.

[48] B. Weisfeiler and A. A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[49] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4602–4609.

[50] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (gans): A survey," *IEEE Access*, vol. 7, pp. 36 322–36 333, 2019.

[51] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding."

[52] H. Cai, V. W. Zheng, and K. C.-C. Chang, "A comprehensive survey of graph embedding: Problems, techniques, and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1616–1637, 2018.

[53] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015.

[54] T. NVIDIA, "V100 gpu architecture whitepaper," 2017.

[55] S. A. Mojumder, M. S. Louis, Y. Sun, A. K. Ziabari, J. L. Abellán, J. Kim, D. Kaeli, and A. Joshi, "Profiling dnn workloads on a volta-based dgx-1 system," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 122–133.

[56] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of graph neural networks with roc," *Proceedings of Machine Learning and Systems (MLSys)*, pp. 187–198, 2020.

[57] T. Bradley, "Gpu performance analysis and optimisation," *NVIDIA Corporation*, 2012.

[58] O. Villa, M. Stephenson, D. Nellans, and S. W. Keckler, "Nvbit: A dynamic binary instrumentation framework for nvidia gpus," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 372–383.

[59] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[60] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishna, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[61] S. Che, B. M. Beckmann, S. K. Reinhardt, and K. Skadron, "Pannotia: Understanding irregular gpgpu graph applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2013, pp. 185–195.

[62] X. Gong, X. Gong, L. Yu, and D. Kaeli, "Haws: Accelerating gpu wavefront execution through selective out-of-order execution," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 2, pp. 1–22, 2019.

[63] G. S. Murthy, M. Ravishankar, M. M. Baskaran, and P. Sadayappan, "Optimal loop unrolling for gpgpu programs," in *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 2010, pp. 1–11.

[64] J. W. Davidson and S. Jinturkar, "An aggressive approach to loop unrolling," Citeseer, Tech. Rep.

[65] X. Xie, Y. Liang, G. Sun, and D. Chen, "An efficient compiler framework for cache bypassing on gpus," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2013, pp. 516–523.

[66] X. Xie, Y. Liang, Y. Wang, G. Sun, and T. Wang, "Coordinated static and dynamic cache bypassing for gpus," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 76–88.

[67] M. K. Tavana, Y. Sun, N. B. Agostini, and D. Kaeli, "Exploiting adaptive data compression to improve performance and energy-efficiency of compute workloads in multi-gpu systems," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 664–674.

[68] L. Ma, Z. Yang, Y. Miao, J. Xue, M. Wu, L. Zhou, and Y. Dai, "Neugraph: parallel deep neural network computation on large graphs," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019, pp. 443–458.

[69] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[70] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, vol. 127, 2012.

[71] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, 2010, pp. 63–74.

[72] L.-N. Pouchet *et al.*, "Polybench: The polyhedral benchmark suite," *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[73] Y. Ukidave, F. N. Paravecino, L. Yu, C. Kalra, A. Momeni, Z. Chen, N. Materise, B. Daley, P. Mistry, and D. Kaeli, "Nupar: A benchmark suite for modern gpu architectures," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, 2015, pp. 253–264.

[74] M. Kulkarni, M. Burtscher, C. Casçaval, and K. Pingali, "Lonestar: A suite of parallel irregular programs," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2009, pp. 65–76.

[75] Y. Sun, X. Gong, A. K. Ziabari, L. Yu, X. Li, S. Mukherjee, C. McCardwell, A. Villegas, and D. Kaeli, "Hetero-mark, a benchmark suite for cpu-gpu collaborative computing," in *2016 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016, pp. 1–10.

[76] A. Li, S. L. Song, J. Chen, X. Liu, N. Tallent, and K. Barker, "Tartan: evaluating modern gpu interconnect via a multi-gpu benchmark suite," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 191–202.

[77] S. Narang and G. Diamos, "Deepbench," 2016.

[78] W. Gao, F. Tang, L. Wang, J. Zhan, C. Lan, C. Luo, Y. Huang, C. Zheng, J. Dai, Z. Cao *et al.*, "Aibench: an industry standard internet service ai benchmark suite," *arXiv preprint arXiv:1908.08998*, 2019.

[79] P. Mattson, V. J. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang *et al.*, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, no. 2, pp. 8–16, 2020.

[80] S. Dong, X. Gong, Y. Sun, T. Baruah, and D. Kaeli, "Characterizing the microarchitectural implications of a convolutional neural network (cnn) execution on gpus," in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 96–106.

[81] K. Kiningham, C. Re, and P. Levis, "Grip: A graph neural network acceleratorarchitecture," *arXiv preprint arXiv:2007.13828*, 2020.

[82] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[83] T. Wang, R. Liao, J. Ba, and S. Fidler, "Nervenet: Learning structured policy with graph neural networks," in *International Conference on Learning Representations*, 2018.