2022 IEEE International Symposium on Workload Characterization (IISWC) | 978-1-6654-8798-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/IISWC55918.2022.00022

gSuite: A Flexible and Framework Independent Benchmark Suite for Graph Neural Network Inference on GPUs

Taha Tekdoğan[1,2] Serkan Göktaş[1] Ayse Yilmazer-Metin[1]

[1]*Department of Computer Engineering, Istanbul Technical University*
[2]*Radar and Electronic Warfare Intelligence Systems, ASELSAN Inc.*

Abstract

As the interest to Graph Neural Networks (GNNs) is growing, the importance of benchmarking and performance characterization studies of GNNs is increasing. So far, we have seen many studies that investigate and present the performance and computational efficiency of GNNs. However, the work done so far has been carried out using a few high-level GNN frameworks. Although these frameworks provide ease of use, they contain too many dependencies to other existing libraries. The layers of implementation details and the dependencies complicate the performance analysis of GNN models that are built on top of these frameworks, especially while using architectural simulators. Furthermore, different approaches on GNN computation are generally overlooked in prior characterization studies, and merely one of the common computational models is evaluated. Based on these shortcomings and needs that we observed, we developed a benchmark suite that is framework independent, supporting versatile computational models, easily configurable and can be used with architectural simulators without additional effort.

Our benchmark suite, which we call gSuite, makes use of only hardware vendor's libraries and therefore it is independent of any other frameworks. gSuite enables performing detailed performance characterization studies on GNN Inference using both contemporary GPU profilers and architectural GPU simulators. To illustrate the benefits of our new benchmark suite, we perform a detailed characterization study with a set of well-known GNN models with various datasets; running gSuite both on a real GPU card and a timing-detailed GPU simulator. We also implicate the effect of computational models on performance. We use several evaluation metrics to rigorously measure the performance of GNN computation. We make gSuite available to research community and provide all the configuration settings which we used for our evaluation so that all the experiments mentioned in the paper are reproducible.

1. Introduction

Graph structured data are highly preferred in many real-world applications due to their ability of expressing the topology of irregular domains. For instance, graphs are used for representing molecules in chemistry [20], relationships among people in social sciences [51], and connections between brain areas in computational neuroscience [42]. Real-world graph datasets have been scaled to enormous amount of sizes in terms of number of nodes, edges, and their feature lengths. Processing these huge-sized data requires an intensive computation. Utilizing Graphics Processing Units (GPUs) is the de facto method in order to meet computation requirements of the graph operations.

Successful application of deep learning techniques in many areas has triggered the idea of applying deep neural network(DNN)-based techniques on the graph structured data. Graph Neural Networks (GNNs) are deep learning based methods that are capable of working on non-euclidean data. GNNs provide a way of performing node level, edge level, and graph level prediction for graph structured data.

There have been various approaches to carry out GNN operations such as message passing (MP) [20, 21] and sparse matrix multiplication (SpMM) [1]. Therefore, GNN computation can be applied in various ways. The increasing number of GNN research motivated developers to extend commonly used deep learning frameworks to support GNN operations using MP or SpMM computational models.

The most popular GNN frameworks are built on top of the commonly used Python-based deep learning frameworks (e.g., PyTorch Geometric (PyG) [18] is built on PyTorch; Deep Graph Library (DGL) [61] gives end user a choice to alternate among PyTorch, Tensorflow or MXNet). Even though these frameworks provide ease of development, they bring dependency to implementations within the base-lined framework and the underlying development libraries.

The increasing number of studies in GNN area has led the benchmarking studies to evaluate the performance of GNN computations. In Table 3, we summarize the main frameworks and benchmarking studies on GNNs. As the table shows, all of the existing frameworks and benchmarking studies utilize at least one of the existing DNN/GNN frameworks and their libraries. The dependencies to the existing frameworks rather complicates the performance analysis and characterization of GNN computations. Most of the computer architecture studies favor utilizing detailed architectural simulators. However, the dependency chain of the existing frameworks makes performance analysis of the GNN applications inaccessible, especially while using architectural simulators.

While the GNN frameworks are intended to be extendable, benchmarks and characterization studies have been only on a limited number of well known GNN models and datasets [4, 66, 72]. Additionally, most of these frameworks

978-1-6654-8798-6/22/$31.00 ©2022 IEEE
DOI 10.1109/IISWC55918.2022.00022

and benchmarks are based on a specific computational model.

All of these limitations of the existing studies and efforts motivated us to develop a configurable and framework independent benchmark suite for GNN Inference. In this paper, we introduce our GNN benchmark suite which we call *gSuite*. gSuite is highly flexible, allows either using an existing framework (such as PyG or DGL) or using our GNN implementations that only make use of the hardware vendor's libraries. The parameters of the desired GNN pipeline, such as the GNN model, the dataset, the number of GNN layers, etc., can be easily configured by passing a few parameters to the program. We built gSuite as the collection of utilities (data import, transform, etc.) and core kernels which are the most primitive operations of GNNs. Therefore, it is extendable to create new GNN models and study their performance on GPUs. gSuite is designed for efficiently studying the performance of GNNs with either hardware profilers or cycle accurate simulators. It does not require additional effort to utilize an architectural simulator, which makes GNN-related operations quite accessible in terms of performance characterization. As a proof of concept, in this work, we characterize the most popular GNN models on varying datasets. We interpret the experimental results in terms of the effect of input workload, GNN model and computational model.

In summary, in this study, we make the following contributions:

- We provide a flexible and user-friendly benchmark suite for GNN inference, hence a desired GNN pipeline can be easily built by passing only a few parameters.
- We eliminate the dependency of GNN pipelines to other frameworks (such as PyG and DGL) and machine learning utilities (such as PyTorch and Tensorflow).
- We characterize the computation of the most representative GNN models at inference level by comparing two predominant computational models, i.e., message passing (MP) and sparse matrix multiplication (SpMM).
- We demonstrate the accessibility of GNN performance on gSuite using both a hardware profiler and an architectural simulator.
- We present and summarize our performance results and make architectural suggestions based on our findings.

Remaining of this paper is structured as follows: Section 2 introduces fundamentals of GNNs with their notations, formulas, common data formats, prevalent datasets, and popular GNN frameworks. In Section 3, we discuss the state-of-the-art of benchmarking methodologies and characterization studies for GNNs. Section 4 explains our architectural model by declaring core kernels as the most primitive GNN operations. And finally in Section 5, we deliver our benchmarking methodology for evaluating the performance of the GNN computation. Then we deliver our results and discuss them in detail.

2. A Brief Background on GNNs

GNNs were first introduced by Scarselli et al. [54], and many new GNN models were proposed since then [10, 41, 59, 70]. A large set of domains leverage the capability of GNNs [20, 42, 51]. Adopting GNNS to wide range of domains gives rise to many new GNN models with various characteristics. Below, we provide a brief background on GNNs by introducing common notation, the computational approaches and popular frameworks to implement the required GNN operations, widely used graph datasets and graph formats utilized to express them.

Graphs are widely used fundamental data structures that are very successful at expressing real-world data that includes relationships between its entities. A graph $G = (V, E)$ is defined by a set of nodes V, and a set of edges E. Two nodes are neighbours if they are directly connected to each other with an edge. The set of neighbour nodes of a node v is represented as $N(v)$. Nodes may carry a list of features that are represented with a latent vector which holds information, known as *node embedding* in GNN literature. We represent the node embedding of a node v as h_v.

GNN pipelines generally consist of multiple GNN layers L. We denote a specific node embedding for layer k as $h_v^{(k)}$, where $k \epsilon [1, L]$ stands for current GNN layer. In some cases, edges may carry information which is called an *edge embedding*, and it is represented as g_e. Most often, the task of a GNN model is predicting or generating the node or edge embeddings.

Furthermore, node and the edge embeddings can be represented with matrices, instead of latent vectors. Feature information of the vertices in a graph can be represented with a feature matrix X in shape $[|V|, f]$, where f stands for the feature size. The connectivity information between nodes in a graph can also be represented with an adjacency matrix A in shape $[|V|, |V|]$.

We will be using these notations for explaining the mathematical expressions of a core set of GNN models that are widely used and also implemented in our benchmark suite. Table 1 summarizes this notation that we use during our study.

2.1. Computation of GNNs

GNN computation can be evaluated under two major GNN phases: inference and training. Inference phase refers to updating each node embedding in a graph analogous to corresponding GNN schema and pre-trained model coefficients. Training phase refers to optimizing the coefficients of the model. Here in this work, we mainly focus on the inference stage of GNNs. Therefore when we invoke the GNN computation, we imply the computation of GNN inference during the study.

A typical GNN includes two types of operations: *aggregation* (or *message* in some cases [18, 20, 61]) and *combination* (or *update* in some cases [6, 18, 20, 43, 61]). *Aggregation*

TABLE 1: Notations for graph neural networks

Notation	Description
$G(V,E)$	Graph
V	Set of nodes of the graph
$\|V\|$	Number of nodes in the graph
E	Set of edges of the graph
v	A single node where $v \in V$
e	A single edge where $e \in E$
k	Current GNN layer where $k \in [1,L]$
$h_v^{(k)}$	Feature representation of node v at layer k
$g_e^{(k)}$	Feature representation of edge e at layer k
$N(v)$	Neighbourhood nodes of the node v
A	Adjacency matrix of the graph
$X^{(k)}$	Feature matrix of the graph at layer k

TABLE 2: Core MP and SpMM kernels.

Kernel Name	Computational Model	Short Form	Description
indexSelect	MP	is	Indexes the input along specified dimension by using index entries.
scatter	MP	sc	Reduces given input based-on index vector using entries.
sgemm	SpMM /GEMM	sg	Generalized matrix multiplication of two given matrices.
SpGEMM	SpMM /GEMM	sp	Matrix multiplication of two sparse matrices.

refers to capturing information from a node's neighbour nodes and accumulating them into its feature representation. It is done by a predefined aggregator function such as *sum*, *mean*, and *max*. *Combination* refers to updating a node's representation by using the output of aggregation phase, which is mostly a multilayer perceptron (MLP) [52]. Aggregation and combination operations are applied analogous to definition of the corresponding GNN model. Application of these operations forms the implementation of the mathematical definitions of GNN models.

Aggregation and combination operations can be applied to graph datasets based on two classes of computational models: *Message Passing* (MP) [20, 21] and *Sparse Matrix Multiplication* (SpMM) [61]. *MP* model is based on a computation pattern where connected nodes scatter their attributes through neighbourhood nodes (aggregation) and each node updates its node-embedding by using such neighborhood nodes' features (combination). On the other hand, *SpMM* model refers to applying aggregation and combination schemes by reducing them into a sequence of matrix multiplication operations.

2.2. GNN Frameworks

There are a number of frameworks to provide an infrastructure to build and run GNNs pipelines such as PyTorch Geometric [18], Deep Graph Library [61], Graph Nets [5], and Spektral [24]. PyTorch Geometric (PyG) and Deep Graph Library (DGL) are the most popular ones among all these frameworks. PyG is built on top of PyTorch library, and all the implemented GNN models are inherited from a base class called *MessagePassing*. On the other hand, DGL implements GNN models based on SpMM computational model. It gives user a choice to alternate between three frameworks (PyTorch, Tensorflow and MXNet).

We examined widely used GNN frameworks and their model implementations. Then we imitated the MP kernels from PyG and SpMM kernels from DGL to implement core kernels of GNNs. We also purified these kernels from dependencies to other libraries (such as PyTorch). Table 2 provides the list of identified core MP and SpMM kernels.

MP models generally consist of neighbour node calculation (*indexSelect*), scattering the node embedding through these connections (*scatter*), and updating self node embedding with a linear function (*sgemm*). On the other hand, SpMM models consist of a consecutive execution of matrix multiplication operations (*SpGEMM* and *sgemm*). These core kernels are organized to comply corresponding GNN model's computation formula.

2.3. GNN Models

While it is very easy to extend our benchmark suite to include any type of GNN model, we have chosen three widely-used GNN models to implement and base our discussions in this paper. We demonstrate the implementation and detailed performance characterization of these three GNN models using gSuite. These three GNN models are *Graph Convolutional Network (GCN)* [38], *Graph Isomorphism Network (GIN)* [64], and *GraphSAGE (SAG)* [27]. Using the notation that we presented above, we continue providing implementation details of these three widely-used GNN models. Then we explain the implementation of the core kernels of GNNs by mapping the formulas with computational models.

2.3.1. Graph Convolutional Networks.

Graph Convolutional Network (GCN) is a semi-supervised classification method that is an efficient variant of convolutional neural networks designed to operate on graphs [38]. It is motivated by the idea of using layer-wise propagation on graph structured data. GCN is capable of encoding both node features and graph structure with their proposed graph modeling approach. Therefore, it is quite popular in a wide range of implementations from knowledge embedding [68] to face clustering [56].

We can express the GCN computation using both *MP* and *SpMM* computational model. The message passing formula for updating each node embedding of a graph in GCN is given by (1).

$$h_v^{(k+1)} = \Theta\left(\sum_{u \in N(v) \cup \{v\}} \frac{1}{\sqrt{d_u d_v}} h_u \right) \tag{1}$$

In (1), ${h_v}^{(k+1)}$ is the feature representation of the updated node v in $(k+1)^{th}$ layer. h_u is a neighbour node embedding from the set $u \epsilon N(v) \cup \{v\}$. d_v represents the node degree of node v, i.e. the number of edges connected to node v. Θ is a linear activation function.

The formulation of GCN using SpMM model is given in (2).

$$X^{k+1} = \hat{D}^{-1/2}\hat{A}\hat{D}^{-1/2}X^k\Theta \tag{2}$$

In (2), $X^{(k+1)}$ represents the feature matrix of a graph at layer $(k+1)$. $\hat{A}$ is the adjacency matrix with self-loops inserted, i.e.:

$$\hat{A} = A + I.$$

$\hat{D}$ is $\hat{A}$'s diagonal matrix, i.e.:

$$\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}.$$

And finally, Θ is an activation function such as a Rectified Linear Unit (ReLU) [22] or a Sigmoid function [50].

2.3.2. Graph Isomorphism Networks. *Graph Isomorphism Networks (GINs)* combine the discriminative power of Weisfeiler-Lehman (WL) graph isomorphism test [63, 64] with GNN's recursive neighbourhood aggregation by making aggregation phase highly expressive and modeling injective functions. GINs are mostly used for classification and discrimination tasks on graphs [3]. Following formulas explain how node embeddings are updated using *MP* and *SpMM* computational approaches for implementing GINs. (3) shows the MP formula of a single GIN layer, and (4) shows the matrix multiplication version of a GIN layer computation.

$$h_v^{(k)} = \Theta^{(k)}\left((1+\epsilon^{(k)}) * h_v^{(k-1)} + \sum_{u \epsilon N(v)} h_u^{(k-1)}\right) \tag{3}$$

In (3), $h_v^{(k)}$ represents the feature vector of the node v at layer k. $h_u^{(k-1)}$ is the feature representation of a neighbourhood node u at layer $(k-1)$. ε is a constant, and $\Theta^{(k)}$ is an activation function at layer k.

$$X^{k+1} = \Theta^{(k)}\left(\left(A^k + (1+\epsilon) \cdot I\right) \cdot X^k\right) \tag{4}$$

In (4), X^{k+1} represents the feature matrix of a graph. A is an adjacency matrix, I is an identity matrix, ϵ is a constant, and $\Theta^{(k)}$ is an activation function at layer k.

2.3.3. GraphSAGE. *GraphSAGE (SAG)* is a general inductive model which generates previously unseen nodes in a graph by leveraging the current node information [27]. SAG uses aggregating functions to aggregate feature information from node's local neighborhood, instead of training a distinct embedding vector for each node. Even though we could not find an available SpMM version of SAG, we implemented it using only the MP computational model due to its popularity with unsupervised learning on graph structured data [11, 71].

Equation (5) shows the formula of MP-oriented SAG model.

$$h_v^{(k)} = W_1 h_v^{(k-1)} + W_2 * mean_{j \epsilon N(v) \cup \{v\}} h_u \tag{5}$$

In (5), $h_v^{(k)}$ shows the feature representation of a node v at layer k. W_1 and W_2 are scalar weights for self nodes and neighbour nodes, respectively.

2.4. Datasets and Widely Used Graph Formats

In prior GNN studies, we often see datasets that consist of two parts: connectivity information to represent edges in graphs, and content information to embody node embeddings. Frameworks construct graphs in terms of their utilized graph formats by inferring information from these datasets. The most popular graph datasets in GNN studies are Cora, Citeseer [47], Pubmed [53], Reddit [28] and LiveJournal [2, 39].

Graph datasets are generally transformed to one of the following formats to be processed by graph libraries: dense matrix, sparse matrix, coordinate format (COO) and compressed sparse row (CSR).

Dense and sparse matrices are often used as input to graph operations based on matrix multiplication, such as SpMM. On the other hand, COO and CSR formats are compressed formats of the graphs that represent attributes and topology of the graph in low-dimensional vector. These types of graph data formats are commonly used in MP-based frameworks, such as PyG. We include all of these formats in our work, and provide utilities to transform a dataset from one format to another.

3. Limitations of Existing GNN Frameworks and Benchmarking Efforts

As the GNNs are finding application in many areas; several new frameworks, performance analysis and characterization studies have emerged. We review the prior GNN frameworks, benchmarks and characterization studies in chronological order and evaluate them in terms of configurability, framework dependency, model and dataset versatility. A comparison table Table 3 is provided to show existing studies' capabilities in terms of measuring GNN performance and ease of use.

As the Table 3 summarizes, prior works lack one or more of the attributes that is desired for studying the performance characteristics of GNN applications. All of these studies utilize an existing DNN/GNN framework and have layers of dependency chain. Such dependency may decrease the accessibility of GNN performance, especially while utilizing an architectural GPU simulator.

Furthermore, most of the frameworks, benchmarks and characterization studies assume that there merely exists a single computational approach. For instance, Pytorch Geometric (PyG) [18] follows a MP schema as a base class to whole GNN models. On the other hand, Deep Graph Library (DGL) [61] considers GNN computation as an SpMM problem. Benchmarks and characterizations studies

TABLE 3: Summary of the prior GNN frameworks, benchmarks, characterization studies and gSuite with their properties.

Study Name	GNN Models	Frameworks	Datasets	Extendibility	GNN Scope
Pytorch Geometric [18]	GCN, SAG, GIN, RGCN, ...	Pytorch	Cora, CiteSeer, Pubmed, MUTAG, PROTEINS, ...	Yes	Both
Deep Graph Library [61]	GCN, GAT, SAG, GIN, SGC, ...	Pytorch, MXNet, Tensorflow	REDDIT, ARXIV, PROTEINS, ...	Yes	Both
GCN-GPU Characterization [66]	GCN, GIN, SAG	PyG	Cora, CiteSeer, Pubmed, Reddit, LiveJournal	No	Inference
GNN-GPU Characterization [72]	GCN, GAT, GGNN,	PyG, DGL	Cora, CiteSeer, Pubmed, AIFB, MUTAG, BGS	No	Inference
GNNMark [4]	PinSAGE, STGCN, DGCN, GW, KGNN, ARGA, TLSTM	PyG	Cora, CiteSeer, Pubmed, NWP, MVL, LA, PEMS	No	Training
HyGCN [67]	GCN, SAG, GIN	PyG	IMDB, Cora, Citeseer, Colab, Pubmed, Reddit	No	Inference
GRIP [37]	GCN, GIN, G-GCN, SAG	GReTa	Pokec, YouTube LiveJournal, Reddit	No	Inference
gSuite	**GCN, GIN, SAG**	**None**	**Cora, Citeseer, Pubmed Reddit, LiveJournal**	**Yes**	**Inference**

generally utilize one of these frameworks to build GNN pipelines. As a result, such assumption on computational model may limit or lead to wrong conclusions when studying performance characteristics of the workloads.

Moreover, except the GNN development frameworks, the rest of the studies are not extendable. One cannot create a new model or add a specific dataset.

With this study, we identify the need for a benchmark suite that does not limit the users to perform a thorough architectural performance analysis study.

4. gSuite and Our Design Approach

While developing our GNN benchmark suite, we considered three key features: (1) Flexibility, (2) Extendability, and (3) Independence. These features are explained below to point out the cornerstones of our benchmark suite's design approach.

- gSuite is a collection of utility functions (e.g. functions to allow input/output, setting configuration, etc.) and core kernels of MP and SpMM computational models. It is flexible to allow building GNN pipelines by selecting the desired dataset, GNN model, number of layers, computational model, and framework (using either gSuite's core kernels or other framework's implementations).
- gSuite allows researchers and engineers to extend the suite in any direction. By utilizing MP and SpMM core kernels, a new GNN model can be built in a plug-and-play manner.
- gSuite's core kernels do not have any dependency on any GNN/DNN frameworks. However, we still give a choice to the end user for alternating between a GNN framework (PyG or DGL) and our GNN implementations.

4.1. Software Architecture

gSuite provides an interface that enables researchers and engineers to easily build a desired GNN pipeline in a plug-and-play manner. We abstract the usage of our benchmark suite from its code implementation to avoid the intervention of end users from coding. The architecture of underlying software is illustrated in Fig. 1.

When running gSuite, user parameters (e.g. number of layers, GNN model, dataset) are passed to the *User Interface*. These parameters are interpreted by the interface and then passed to the *Abstraction Module*. Nevertheless, the interface does not require the end user to pass all the parameters to the suite. There is a configuration file that includes all these settings as default parameters, where these default

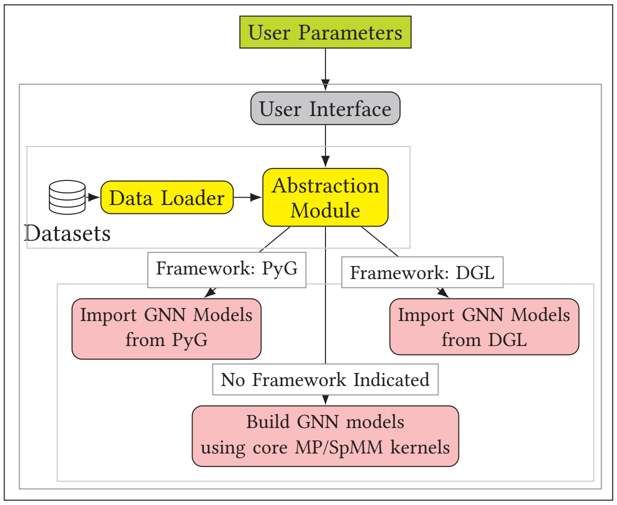

Figure 1: Software architecture of gSuite.

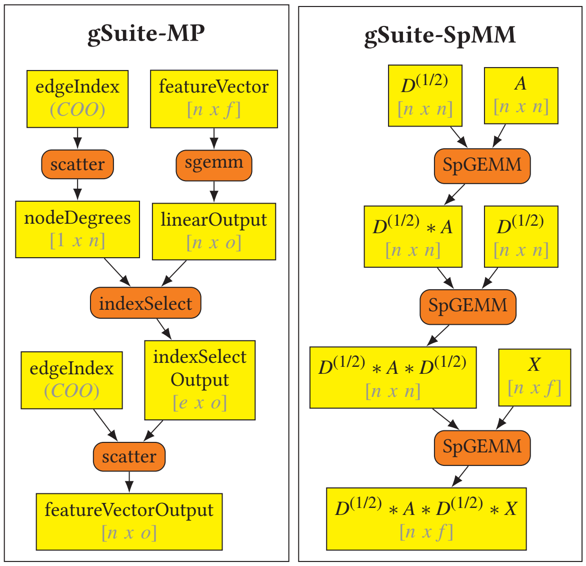

Figure 2: Computational schema of the GCN pipelines of gSuite-MP and gSuite-SpMM. Yellow boxes represent data, orange ones represent the *core kernels*.

TABLE 4: The Included Datasets in our Evaluation

Dataset	Nodes	Feature Length	Edges	Short Form
Cora [47]	2,708	1,433	5,429	CR
CiteSeer [47]	3,327	3,703	4,732	CS
PubMed [53]	19,717	500	44,438	PB
Reddit [28]	232,965	602	11,606,919	RD
LiveJournal [2]	4,847,571	1	68,993,773	LJ

parameters take action when a parameter value is not specified by the user.

The decision of which framework, GNN model and dataset are going to be used is made by this abstraction module. In case of no framework is indicated by the end user; then our GNN implementations are utilized.

Data loader imports the chosen dataset and handles pre-processing stage of GNN computation, i.e., loads edge index vector and feature representation vector.

gSuite implements both of the computation models (MP and SpMM) for each deployed GNN model. Iterative execution of these kernels with proper data manipulation results in a GNN model. To illustrate the phenomenon, graph convolutional network (GCN) inference is implemented by adding *indexSelect*, *scatter* and *linear* kernels to satisfy GCN's MP computation scheme. On the other hand, the SpMM computation of GCN refers to reducing all the above-mentioned operations in a single matrix multiplication. We implemented SpMM GCN by utilizing NVIDIA's cuBlas utilities. An illustration of these implementations are given at Fig. 2.

All the implemented kernels are listed in Table 2 with their brief description. These kernels are designed to be generic and GNN-oriented so that any GNN model can be built by utilizing these kernels.

5. Evaluation

In this section, we first explain the GNN models and datasets that we deployed in our suite. Next, our experimental setup is briefly described. Finally, we deliver the results of our experiments and discuss our observations.

5.1. GNN Models

We implemented the most potent GNN models in our benchmark suite: GCN, GIN and SAG. We made the model implementations two-sided for computational model versatility, i.e., each model has distinct MP and SpMM implementations (except SAG).

MP models consist of *indexSelect*, *scatter*, and *sgemm* kernels; SpMM models incorporate *SpMM* and *sgemm* kernels.

5.2. Datasets

In our evaluation, we used the most prevalent graph datasets across varying domains: Cora, Citeseer [47], Pubmed [53], Reddit [28], and LiveJournal [2, 39]. These datasets vary greatly in terms of size, feature length, number of nodes and edges. A table of datasets with their information is given in Table 4.

Each of these datasets represents a particular feature that aims to test the limitations of implemented GNN models and underlying architectures. Each attribute of the datasets makes them unique in terms of computation. For instance, one may include a large amount of feature size; while other may have a huge number of directed edges.

5.3. Experimental Setup

Our experiments were conducted on NVIDIA V100 GPU 32GB and Intel Xeon 2000 CPU. Each GNN model with the specified input set is run three times; and the mean values of the statistics of these runs were collected. Profiling operations are done at the kernel level for all GNN pipelines.

NVIDIA's nvprof [7] is used for collecting the results on the GPU card. It is a profiling tool that tracks running applications on GPUs and collects information about the performance activity of the application. We use the version 10.2 of nvprof.

GPGPU-Sim [40] is utilized for collecting more detailed performance statistics. GPGPU-Sim is a timing detailed architectural simulator which is capable of running CUDA and OpenCL kernels. We use the configuration file that is provided by the simulator package and models a GPU architecture similar to NVIDIA's V100 GPU. We used the version 4.0 of GPGPU-Sim.

5.4. Results

5.4.1. Execution Time. We start our evaluation by measuring execution time of GNN pipelines; using their implementations with PyG, DGL, gSuite-MP, and gSuite-SpMM. We measure the execution time as wall clock time. In general, execution times of PyG is longer than other frameworks. This is mainly because of the initializations that are performed as part of their implementation. Since the gSuite eliminates high level library dependencies, its implementations tend to run faster than other frameworks in terms of end-to-end execution time. We compare these durations in Fig. 3.

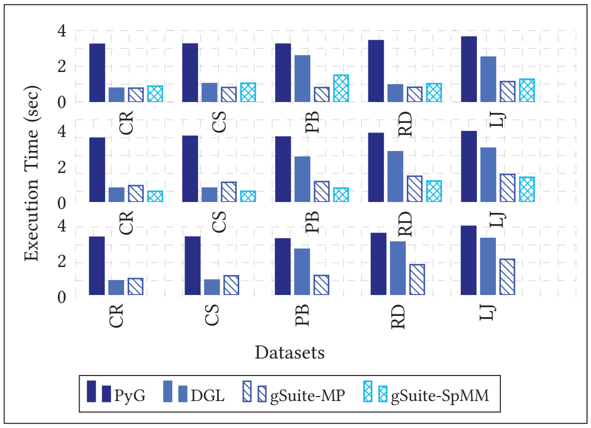

Figure 3: End-to-end execution time of frameworks with different GNN models on varying datasets.

We also show the execution time distribution of the kernels in Fig. 4. gSuite shows a similar distribution to that of PyG and DGL. We observed that the GNN model is the main determinative factor for the distribution of kernel execution times.

5.4.2. Instruction Breakdown. Each core kernel consists of different types of instructions to accomplish its task during the execution. We have found that each core kernel has a characteristic distribution of instructions that does not vary even though when GNN model or dataset is adjusted. Fig 5 shows the instruction breakdown of kernels on different models and datasets, implying that the distribution is not affected by the adjustment of GNN model and dataset.

From our instruction breakdown analysis (Fig. 5), we observe that scatter and indexSelect kernels are dominated with integer operations. Because, these two kernels mainly perform address calculations for data accesses. On the other hand, sgemm kernel is highly dominated by floating operations. Based on these observations, we can suggest researchers to investigate co-scheduling of kernels and also focus on warp scheduling studies for better utilization of the functional units.

5.4.3. Issue Stall Distribution. We evaluate and analyze the issue stall distribution of core GNN kernels. Issue stalls explain why an active warp is not eligible during its execution. Prior studies showed that the change in the characteristics of input workload has a strong effect on GNN computation [66, 72]. We observe a similar behaviour in our experiments. As the size of the dataset gets larger, all the core kernels except sgemm develop memory dependency. Fig. 6 illustrates the issue stall cycle distribution of core kernels in MP and SpMM based implementations of GCN, GIN SAG models, running with our five datasets.

We found that memory dependency is the dominant stall in both MP- and SpMM-based implementations, with 46.3% on average. This is due to irregular memory access pattern of GNN Inference tasks.

5.4.4. Warp Occupancy Distribution. This metric stands for the ratio of active warps to maximum number of supported active warps, from GPGPU-Sim. We use this metric to measure the utilization of functional units. In this analysis, *stall* state shows that pipeline is stalled and therefore cannot issue any instructions. *Idle* state means the warps were issued but not ready to execute next instruction. Finally, *WX* refers to *X* active threads were scheduled into pipeline.

During the experiments, we observed that the type of GNN model plays a crucial role in pipeline utilization. MP-based kernels (scatter and indexSelect) of GCN tend to stay idle during the execution, unlike GIN and SAG kernels. However, sgemm kernel is immune to these GNN model adjustments. Fig. 7 shows how the utilization levels change across GNN models and our datasets.

Figures 6 and 7 also highlights the inefficiency of front-end when running indexSelect and scatter kernels in GCN MP model (we observe high instruction fetch in Fig. 6 and high idle time in Fig. 7 for these kernels in GCN MP model, especially with small sized datasets (CR and CS)).

5.4.5. L1/L2 Cache Hit Rate. As GNN operations draw an irregular access pattern on memory; we expect high miss rates consistent with prior characterization studies [4, 55]. We aim to show how input workload characteristics affect the cache miss rates during GNN computation.

Moreover, we point out resemblance and differences of a hardware profiler statistics and architectural simulator

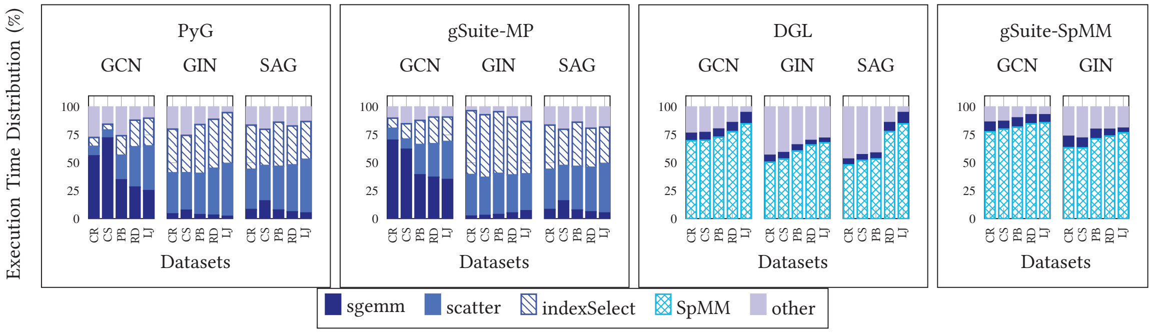

Figure 4: Execution time distribution of the kernels with different GNN frameworks.

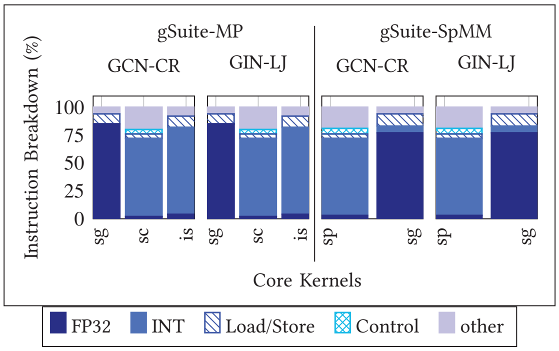

Figure 5: Instruction breakdown of the kernels during the execution.

results for this metric. We collected the cache utilization statistics by using both nvprof and GPGPU-Sim. Fig. 8 depicts the results of experiments and compares nvprof results with GPGPU-Sim outcomes.

We observe that L1 cache hit ratio values for profiler and simulator are more aligned than L2 cache hit values. Specifically, for some workloads (CR and CS), the simulator-indicated memory performance is not well matching with the hardware-based memory performance. This shows us that more validation study is required on GPGPU-Sim's memory model.

From our detailed analysis on L1/L2 cache accesses/hits, we observe that the GCN workloads have some or limited locality. This suggests that architects should study GNN friendly caching and prefetching options. Specifically, indexSelect kernel cannot utilize memory efficiently. Average memory utilization of 34.6% combined with the high L1D cache miss rates we observed, we suggest researchers to investigate other caching techniques to be applied particularly on indexSelect kernel.

We also notice that larger input workloads result in less L1/L2 cache hit ratios. These extremely low L1D cache hit rates points out that caching may not be a good technique for GNN-Inference. Therefore using L1 cache bypassing techniques can be considered as an alternative to alleviate such a problem.

From issue stall distribution (Fig. 6) and L1/L2 cache hit rates (Fig. 8), we observe that indexSelect and scatter kernels suffer from memory dependency. This suggests that considering the implementations of functionalities of these kernels on memory side would be good option in terms of energy consumption, utilization and performance. The atomic reduce operation in scatter kernel affects the performance of this kernel. Therefore, this kernel could benefit from the architectural support for more efficient synchronization operations. Architects may also investigate the prefetching options.

5.4.6. Compute/Memory Utilization. We examine the performance limiter for each core kernel during the execution. Low compute and memory utilization values point that a kernel's performance is bounded by instruction and memory latency. We observe that scatter kernel utilizes memory more efficiently than other kernels, especially when employed in GIN and SAG. Compute and memory utilization of sgemm kernel scales up as the input workload is bigger (e.g. with LiveJournal dataset). These utilization levels are presented in Fig. 9.

6. Related Work

The continuous growth of real-world graph data has led to the development of new methods to process such data effectively [23, 44, 46, 73]. With their ability to handle high memory access bandwidth and massive parallelism, GPUs gained the attention of researchers and engineers, especially for graph processing tasks [1, 30, 49]. While GPUs offer significant performance improvements for graph applications, they come with several challenges to deal with. There are many studies on efficient implementation and performance evaluation of graph applications on GPUs [17, 25, 29, 35, 45, 49, 60, 69]. These studies mainly focus on data layout optimizations, memory access patterns optimizations, and workload mapping for load balancing. There are also several graph frameworks, benchmarks and characterization studies on graph algorithms running on GPUs [8, 31, 36, 62, 65].

Most DNN applications and frameworks also utilize the GPUs' computing capability. We have seen many

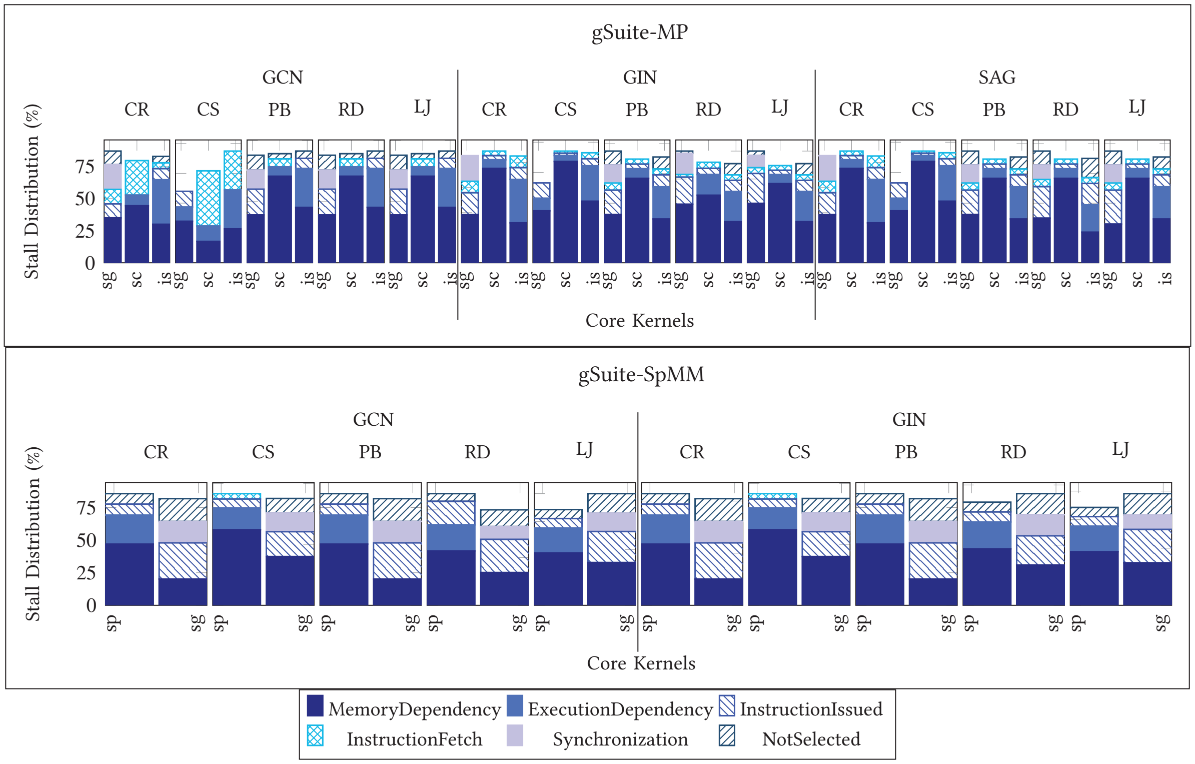

Figure 6: Issue stall distribution of the kernels during the execution, comparing MP and SpMM kernels across different GNN models and datasets.

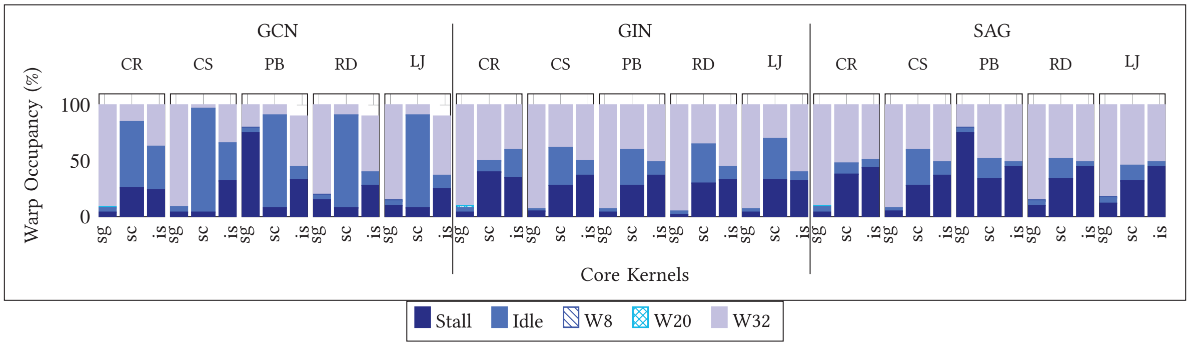

Figure 7: Warp occupancy distribution of the gSuite-MP kernels on varying GNN models and datasets.

DNN frameworks [34, 57] and studies that measure the performance of DNN applications on GPUs [9, 12–14, 26, 32, 33, 74].

Increasing interest in GNNs has led the model benchmarking [15, 16, 19, 48] and architectural performance analysis studies for GNNs [4, 66, 72]. There are also efforts towards providing datasets for benchmarking GNNs [48].

The most relevant work to ours are [4, 18, 37, 61, 66, 67, 72].

PyTorch Geometric (PyG) is a GNN framework based on PyTorch, which provides an infrastructure to build GNN pipelines with implemented GNN models and datasets. GNN models in PyG are inherited from a base class called *MessagePassing*. Another common GNN framework is Deep Graph Library (DGL), which gives user a choice to alternate among three DNN libraries: PyTorch, Tensorflow, and MXNet. DGL follows the *SpMM* schema in its GNN implementations.

Yan et al. [66] characterize GCNs at Inference level with varying workloads, using PyG. Zhang et al. [72] characterize GNN Inference on GPUs by taking two popular frameworks into consideration: PyG and DGL. They consider the most common GNN models in a stage level analysis manner,

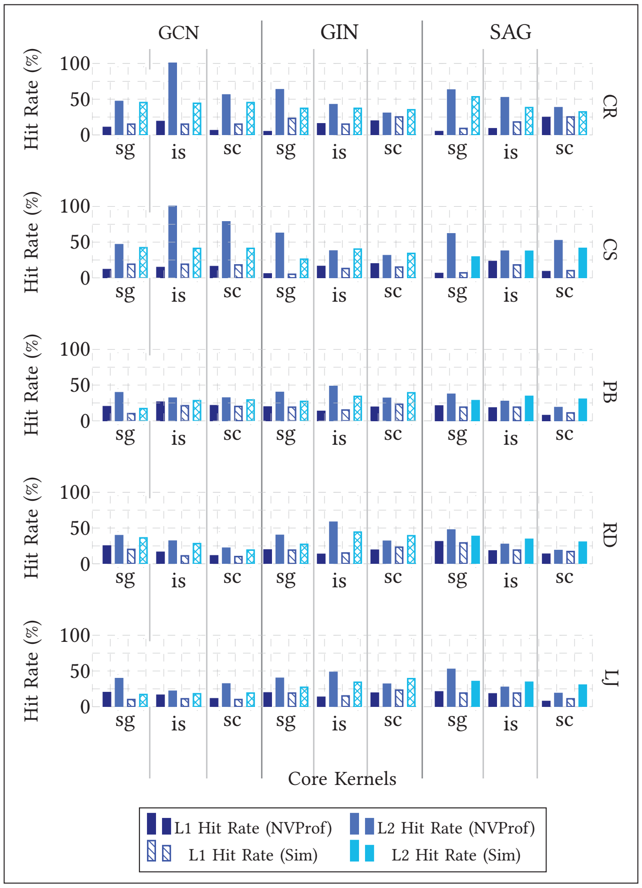

Figure 8: L1 and L2 cache hit rates of the MP-gSuite kernels, comparing NVIDIA Profiler and GPGPU-Sim outcomes.

and make implications for hardware accelerators. However, this work is not open-source and cannot be extended by research community.

GNNMark [4] is a benchmark suite that is designed to understand system-level and architectural implications of GNNs, specifically during the training phase. A range number of GNN models are covered, and many datasets are used. They examine the scalability of GNN training across a multi-GPU system. However, unlike our study, GNNMark is not intended to be configurable. Workloads are tend to be treated as applications of model-dataset couples. GNNMark is also using PyG and DGL to build GNN pipelines.

7. Conclusion and Future Work

In this paper, we present gSuite, a flexible and framework-independent benchmark suite for GNNs. By providing this suite, we aim to fill the absence of a GNN-oriented benchmark utility which is not dependent to any other framework such as PyG and DGL. As a proof of concept, we characterize and profile the computation of GNN Inference by using our proposed benchmark suite.

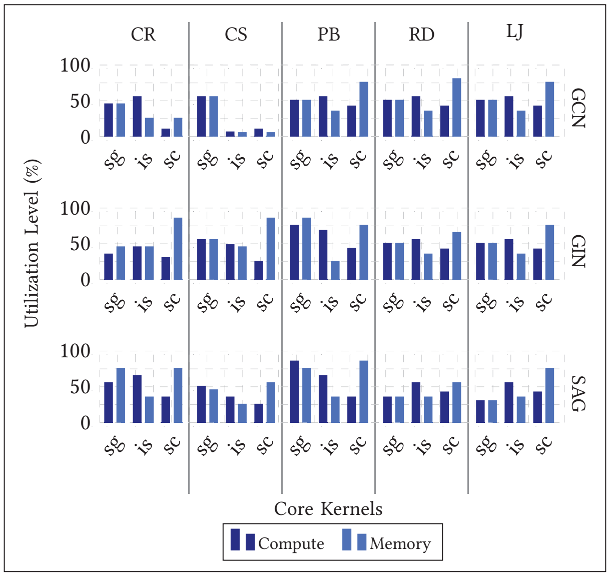

Figure 9: Compute and memory utilization levels of MP-gSuite kernels on varying GNN models and datasets.

We utilize both a hardware profiler and a cycle accurate simulator to measure the performance of GNN computation.

We provide gSuite as an open-source project, hence all the experiments are reproducible with proper configurations [58]. We also welcome any contribution and suggestion to the benchmark suite.

As a future work, we plan to extend our benchmark suite by adding support for GNN-Training, which includes the implementation of training-related aspects such as neuron layers, propagations, weights, etc.

We also plan to support different architectures such as FPGAs and AMD GPUs by implementing our core kernels with OpenCL.

References

[1] A. Ashari, N. Sedaghati, J. Eisenlohr, S. Parthasarath, and P. Sadayappan, “Fast sparse matrix-vector multiplication on gpus for graph applications,” in *SC ’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, pp. 781–792.

[2] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, “Group formation in large social networks: membership, growth, and evolution,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 44–54.

[3] S. Bandyopadhyay, M. Aggarwal, and M. N. Murty, “A deep hybrid pooling architecture for graph classification with hierarchical attention,” in *Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 554–565. [Online]. Available: https://doi.org/10.1007/978-3-030-75762-5_44

[4] T. Baruah, K. Shivdikar, S. Dong, Y. Sun, S. A. Mojumder, K. Jung, J. L. Abellán, Y. Ukidave, A. Joshi, J. Kim, and D. Kaeli, “Gnnmark: A benchmark suite to characterize graph neural network training on gpus,” in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 13–23.

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. 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," 2018. [Online]. Available: https://arxiv.org/abs/1806.01261

[6] 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. M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *CoRR*, vol. abs/1806.01261, 2018. [Online]. Available: http://arxiv.org/abs/1806.01261

[7] T. Bradley, "Gpu analysis and optimisation - people.maths.ox.ac.uk," 2012. [Online]. Available: https://people.maths.ox.ac.uk/~gilesm/cuda/lecs/NV_Profiling_lowres.pdf

[8] 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)*, 2013, pp. 185–195.

[9] 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)*, 2009, pp. 44–54.

[10] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," *CoRR*, vol. abs/1801.10247, 2018. [Online]. Available: http://arxiv.org/abs/1801.10247

[11] Y. Chen, Y. Hu, K. Li, C. K. Yeo, and K. Li, "Approximate personalized propagation for unsupervised embedding in heterogeneous graphs," *Inf. Sci.*, vol. 600, no. C, p. 287–300, jul 2022. [Online]. Available: https://doi.org/10.1016/j.ins.2022.04.002

[12] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[13] C. A. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. D. Bailis, K. Olukotun, C. Ré, and M. A. Zaharia, "Dawnbench : An end-to-end deep learning benchmark and competition," 2017.

[14] S. Dong and D. Kaeli, "Dnnmark: A deep neural network benchmark suite for gpus," in *Proceedings of the General Purpose GPUs*, ser. GPGPU-10. New York, NY, USA: Association for Computing Machinery, 2017, p. 63–72. [Online]. Available: https://doi.org/10.1145/3038228.3038239

[15] V. Dwivedi, C. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," 03 2020.

[16] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[17] B. O. Fagginger Auer and R. H. Bisseling, *A GPU Algorithm for Greedy Graph Matching*. Berlin, Heidelberg: Springer-Verlag, 2012, p. 108–119.

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

[19] V. Fung, J. Zhang, E. Juarez, and B. Sumpter, "Benchmarking graph neural networks for materials chemistry," *ChemRxiv*, 2021.

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

[21] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Message Passing Neural Networks," in *Lecture Notes in Physics, Berlin Springer Verlag*, K. T. Schütt, S. Chmiela, O. A. von Lilienfeld, A. Tkatchenko, K. Tsuda, and K.-R. Müller, Eds., 2020, vol. 968, pp. 199–214.

[22] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323. [Online]. Available: https://proceedings.mlr.press/v15/glorot11a.html

[23] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. USA: USENIX Association, 2012, p. 17–30.

[24] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral [application notes]," *Comp. Intell. Mag.*, vol. 16, no. 1, p. 99–106, feb 2021. [Online]. Available: https://doi.org/10.1109/MCI.2020.3039072

[25] A. V. P. Grosset, P. Zhu, S. Liu, S. Venkatasubramanian, and M. Hall, "Evaluating graph coloring on gpus," in *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 297–298. [Online]. Available: https://doi.org/10.1145/1941553.1941597

[26] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the deployment of deep neural networks on commercial edge devices," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 2019, pp. 35–48.

[27] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[28] W. L. Hamilton, J. Zhang, C. Danescu-Niculescu-Mizil, D. Jurafsky, and J. Leskovec, "Loyalty in online communities," 2017. [Online]. Available: https://arxiv.org/abs/1703.03386

[29] P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the gpu using cuda," in *Proceedings of the 14th International Conference on High Performance Computing*, ser. HiPC'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 197–208.

[30] G. He, H. Feng, C. Li, and H. Chen, "Parallel simrank computation on large graphs with iterative aggregation," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 543–552. [Online]. Available: https://doi.org/10.1145/1835804.1835874

[31] C. Hong, A. Sukumaran-Rajam, J. Kim, and P. Sadayappan, "Multigraph: Efficient graph processing on gpus," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2017, pp. 27–40.

[32] T. Horikawa, S. C. Aoki, M. Tsukamoto, and Y. Kamitani, "Characterization of deep neural network features by decodability from human brain activity," *Scientific data*, vol. 6, no. 1, pp. 1–12, 2019.

[33] S. Huang, W. Peng, Z. Jia, and Z. Tu, "One-pixel signature: Characterizing cnn models for backdoor detection," in *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 326–341. [Online]. Available: https://doi.org/10.1007/978-3-030-58583-9_20

[34] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 675–678. [Online]. Available: https://doi.org/10.1145/2647868.2654889

[35] O. Kalentev, A. Rai, S. Kemnitz, and R. Schneider, "Connected component labeling on a 2d grid using cuda," *Journal of Parallel and Distributed Computing*, vol. 71, no. 4, pp. 615–620, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731510002108

[36] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "Cusha: Vertex-centric graph processing on gpus," in *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 239–252. [Online]. Available: https://doi.org/10.1145/2600212.2600227

[37] K. Kiningham, P. Levis, and C. Ré, "Grip: A graph neural network accelerator architecture," *IEEE Transactions on Computers*, pp. 1–12, 2022.

[38] 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*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[39] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[40] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers, and T. M. Aamodt, "Analyzing machine learning workloads using a detailed gpu simulator," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 151–152.

[41] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[42] X. Li, Y. Zhou, N. C. Dvornek, M. Zhang, J. Zhuang, P. Ventola, and J. S. Duncan, "Pooling regularized graph neural network for fmri biomarker analysis," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2020: 23rd International Conference, Lima, Peru, October 4–8, 2020, Proceedings, Part VII*. Berlin, Heidelberg: Springer-Verlag, 2020, p. 625–635. [Online]. Available: https://doi.org/10.1007/978-3-030-59728-3_61

[43] R. Liao, M. Brockschmidt, D. Tarlow, A. L. Gaunt, R. Urtasun, and R. Zemel, "Graph partition neural networks for semi-supervised classification," 2018. [Online]. Available: https://arxiv.org/abs/1803.06272

[44] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," in *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'10. Arlington, Virginia, USA: AUAI Press, 2010, p. 340–349.

[45] L. Luo, M. Wong, and W.-m. Hwu, "An effective gpu implementation of breadth-first search," in *Proceedings of the 47th Design Automation Conference*, ser. DAC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 52–55. [Online]. Available: https://doi.org/10.1145/1837274.1837289

[46] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 135–146. [Online]. Available: https://doi.org/10.1145/1807167.1807184

[47] A. Mccallum, K. Nigam, and J. Rennie, "Automating the construction of internet portals," 03 2000.

[48] P. Mernyei and C. Cangea, "Wiki-cs: A wikipedia-based benchmark for graph neural networks," *arXiv preprint arXiv:2007.02901*, 2020.

[49] D. Merrill, M. Garland, and A. Grimshaw, "Scalable gpu graph traversal," *SIGPLAN Not.*, vol. 47, no. 8, p. 117–128, feb 2012. [Online]. Available: https://doi.org/10.1145/2370036.2145832

[50] S. Narayan, "The generalized sigmoid activation function: Competitive supervised learning," *Information Sciences*, vol. 99, no. 1, pp. 69–82, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025596002009

[51] D. Nettleton, "Data mining of social networks represented as graphs," *Computer Science Review*, vol. 7, pp. 1–34, 02 2013.

[52] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Trans. Cir. and Sys.*, vol. 8, no. 7, p. 579–588, jul 2009.

[53] R. A. Rossi and N. K. Ahmed, "The network data repository with interactive graph analytics and visualization," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, p. 4292–4293.

[54] 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, 2009.

[55] A. Subramaniyan, Y. Gu, T. Dunn, S. Paul, M. Vasimuddin, S. Misra, D. Blaauw, S. Narayanasamy, and R. Das, "Genomicsbench: A benchmark suite for genomics," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 1–12.

[56] D. Sun, K. Yang, and Z. Ding, "Confidence-based simple graph convolutional networks for face clustering," *IEEE Access*, vol. 10, pp. 6459–6469, 2022.

[57] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint arXiv:1605.02688*, may 2016. [Online]. Available: https://arxiv.org/pdf/1605.02688

[58] T. Tekdogan, "gsuite," Sep. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.7071370

[59] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017. [Online]. Available: https://arxiv.org/abs/1710.10903

[60] V. Vineet, P. Harish, S. Patidar, and P. J. Narayanan, "Fast minimum spanning tree for large graphs on the gpu," in *Proceedings of the Conference on High Performance Graphics 2009*, ser. HPG '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 167–171. [Online]. Available: https://doi.org/10.1145/1572769.1572796

[61] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[62] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the gpu," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2851141.2851145

[63] B. Weisfeiler and A. Leman, "The reduction of a graph to canonical form and the algebra which appears therein," *NTI, Series*, vol. 2, no. 9, pp. 12–16, 1968.

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

[65] Q. Xu, H. Jeon, and M. Annavaram, "Graph processing on gpus: Where are the bottlenecks?" in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 140–149.

[66] 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.

[67] 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)*, 2020, pp. 15–29.

[68] S. Yao, D. Pi, and J. Chen, "Knowledge embedding via hyperbolic skipped graph convolutional networks," *Neurocomputing*, vol. 480, pp. 119–130, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222000546

[69] K. Yonehara and K. Aizawa, "A line-based connected component labeling algorithm using gpus," in *2015 Third International Symposium on Computing and Networking (CANDAR)*, 2015, pp. 341–345.

[70] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

[71] T. Zhang, H.-R. Shan, and M. A. Little, "Causal graphsage: A robust graph method for classification based on causal sampling," *Pattern Recogn.*, vol. 128, no. C, aug 2022. [Online]. Available: https://doi.org/10.1016/j.patcog.2022.108696

[72] 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.

[73] J. Zhou, C. Xu, X. Chen, C. Wang, and X. Zhou, "Mermaid: Integrating vertex-centric with edge-centric for real-world graph processing," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, 2017, pp. 780–783.

[74] 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)*, 2018, pp. 88–100.

Appendix

1. Abstract

This Artifact Appendix introduces the experimental setup of gSuite by guiding how to access, install, compile and execute the experiments described in the paper.

2. Artifact check-list (meta-information)

- **Algorithm:** graph neural network (GNN) inference
- **Compilation:** make (4.1 or higher), nvcc (8.0 or higher)
- **Model:** GCN, GIN, SAG
- **Data set:** Cora, CiteSeer, Pubmed, Reddit, LiveJournal
- **Hardware:** NVIDIA GPU
- **Execution:** `python3 main.py –config "conf.json"`
- **Metrics:**
  - Execution Time
  - LD/ST Instructions
  - Warp Occupancy
  - Issue Stall Distribution
  - L1/L2 Cache Hit Rate
- **Output:** profiler and simulator outputs if used
- **Experiments:** 1-layer GNN pipelines
- **How much disk space required (approximately)?:** 2.6 GB
- **How much time is needed to prepare workflow (approximately)?:** around 5 mins
- **How much time is needed to complete experiments (approximately)?:** 30 mins
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** CCA 4.0 International
- **Data licenses (if publicly available)?:** referenced
- **Workflow framework used?:** No
- **Archived (provide DOI)?:** 10.5281/zenodo.7071370

3. Description

3.1. How to access. Download the stable and up-to-date version of gSuite from `https://zenodo.org/record/7071370` .

3.4. Data sets. Data sets are included in the ‘cuda‘ folder as pairs: ‘content‘ and ‘cites‘. Content incorporates the feature information of nodes of the corresponding graph. Cites incorporates the connectivity information among nodes, i.e. edges. One can easily import a custom dataset into gSuite by following the above-described data set format. Data sets included in gSuite are Cora, Citeseer, PubMed, Reddit, and LiveJournal.

3.2. Hardware dependencies. Any NVIDIA GPU with NVIDIA toolkit version higher than 8.0 is sufficient for executing GNN pipelines on GPUs.

There is no restriction for running experiments on CPUs.

3.3. Software dependencies. CUDA Toolkit 8.0 or higher for GPU kernels. make 4.1 or higher to compile source codes.

3.5. Models. GNN models included in gSuite are Graph Convolutional Network (GCN), Graph Isomorphism Network (GIN), and GraphSAGE (SAG). All of these models are implemented as two-sided in terms of computational model: Message Passing (MP) and Sparse Matrix Multiplication (SpMM).

4. Installation

Download the repository (see Section A.3.1). Extract the material from “gSuite.rar". Add the following environment variables:

```
$ export PATH=CUDA_PATH/bin:$PATH
$ export CPATH=CUDA_PATH/include:$CPATH
$ export LD_LIBRARY_PATH=CUDA_PATH/lib64:$LD_LIBRARY_PATH
```

Then navigate to **cuda** folder and execute **make** command. Executables will be generated in the same folder complying with your current architecture.

5. Experiment workflow

Executing the main script by passing one mandatory (config file) and a few optional parameters (model, dataset, etc.).

```
python3 main.py --config "conf.json"
    --model "gcn" --dataset "cora"
```