MEMORYBRIDGE: Leveraging Cloud Resource Characteristics for Cost-Efficient Disk-based GNN Training via Two-Level Architecture

Yoochan Kim[1], Weikuan Yu[2], Hong-Yeon Kim[3], Youngjae Kim [1,†]

[1]*Dept. of Computer Science and Engineering, Sogang University, Seoul, Republic of Korea*
[2]*Dept. of Computer Science, Florida State University, FL, USA,* [3]*ETRI, Daejeon, Republic of Korea*

***Abstract*—Graph Neural Networks (GNNs) are machine learning models that process graph-structured data by learning relationships between vertices and edges, as well as graph-level characteristics. Recently, with the emergence of large graph datasets on a TB scale, dataset sizes have exceeded the memory capacity of single machines. As a result, traditional methods that load all graph data into memory have become unusable, leading to the emergence of disk-based GNN training that uses storage as a memory extension. Recent research has focused on reducing disk I/O bottlenecks in disk-based GNNs. However, disk-based GNNs face new challenges in cloud environments due to two main characteristics. First, compared to node-local environments, the significantly slower cloud storage I/O speed becomes the main bottleneck of the entire training process. Second, pre-defined virtual machines prevent users from freely utilizing desired memory sizes, bandwidth, and the latest GPU technologies. These limitations have made existing disk-based GNN research unusable in cloud environments. To overcome this, we propose MEMORYBRIDGE, a system that cost-effectively accelerates GNN training in cloud environments through a novel two-level architecture that utilizes affordable GPU resources as training nodes and remote memory resources without GPUs as memory nodes, instead of using a single expensive GPU resource. This architecture consists of two key components: (i) a mathematical solver that recommends the most cost-effective resource combination, and (ii) a cloud-specialized GNN framework that implements graph-aware fixed caching and batch pipelining optimization. The experimental results show that MEMORYBRIDGE achieved a speed improvement of up to 32.7x compared to existing GNN training frameworks and a cost efficiency of 9.9x compared to alternative resource configuration strategies, effectively handling the unique problems that arise from the combination of cloud environments and GNN training.**

***Index Terms*—Graph Neural Network, Cloud Computing, System Architecture, Resource Optimization**

I. INTRODUCTION

Graph Neural Network (GNN) has established itself as an innovative deep learning model capable of processing graph data consisting of nodes and edges to analyze the relationships between nodes and edges, as well as the characteristics of the graph itself [1]. While graph data has traditionally been used in many fields, including e-Commerce, molecular biology [2], social networks [3], and finance, its unstructured nature prevented its use in DNN. Today, GNN is being applied effectively in recommendation systems [4, 5], pharmaceutical field, and anomaly detection based on tasks such as node classification, link prediction, and graph clustering [6, 7, 8].

GNN requires different data sampling compared to traditional approaches. While target data for traditional DNNs, such as text corpora, images, and sounds, consisted of independent data points, graph data points (nodes, edges) are interconnected. Therefore, to generate batches, one must not only explore target data points, but also search their neighborhoods and collect feature vectors corresponding to the embeddings of the explored data. This process requires extensive data access. To reduce data access costs, early GNN frameworks chose to load the entire dataset into memory and maintain it during training. However, as graph data sizes grew larger, issues arose where they exceeded single-machine memory capacity, necessitating expansion beyond single machine memory capacity. Recent research has adopted disk-based GNN methodology, which stores graph data on disk and accesses it during training, focusing on reducing data access costs. Instead of memory expansion through distributed learning, expansion using affordable and high-capacity disks like SSDs was deemed more economical.

Meanwhile, cloud computing, a service that provides computing resources via the Internet, has become a good infrastructure for traditional DNN training due to its advantages of flexibility, scalability, and ease of initial infrastructure setup. However, cloud storage resources have unique characteristics.

First, unlike traditional node-local computing, cloud storage is provided in a separated form connected through networks. While this design allows flexible expansion of the required capacity, it results in performance degradation due to network communication overhead. For example, while node-local NVMe SSDs provide bandwidth of 1,500 MB/s - 3,500 MB/s, general-purpose cloud storage (e.g., AWS GP2) is limited to maximum 250 MB/s, showing about 14 times performance difference, and cloud storage latency is also about 3-4 times higher than local storage. This performance gap significantly degrades the performance of disk-based GNN, which is sensitive to disk I/O, and nullifies existing disk-based GNN solutions.

Second, pre-defined virtual machines mostly do not support

[†]Y. Kim is the corresponding author.

latest technologies like GPU-based I/O, and users cannot combine desired computing resources. For example, combinations such as large memory, high storage bandwidth, modest CPU, and cheap GPU are not available. This, coupled with the first characteristic, makes it even more difficult for users to utilize resources as desired.

Therefore, users performing GNN training in cloud environments face issues where they cannot use as many instances as desired, and even if used, cannot use them at cost-effective prices.

Four Observations about GNN and Cloud. To perform disk-based GNN in cloud environments, resources that are both cost-effective and sufficiently useful for disk-based GNN must be utilized. To achieve this, we derived the following four observations from cloud environments and GNN training. First, resource performance and cost in cloud environments are not necessarily proportional. In particular, computing resources with GPUs require relatively higher costs due to high demand. In contrast, resources without GPUs are provided at several times lower prices despite having more CPU cores, memory capacity, and bandwidth than GPU resources. Second, GNN does not require intensive GPU computation, allowing the utilization of lower-performance GPUs compared to other machine learning models. Instead, disk-based GNN is highly dependent on storage bandwidth, speed, and number of CPU cores. In particular, when the storage bandwidth is low, the sampling time for batch generation increases exponentially, becoming a major bottleneck. Third, computing resources specialized for disk-based GNN in cloud environments are not cost-effective. GPU-equipped computing resources are excessively priced compared to their GPU-less counterparts. Fourth, network bandwidth of computing resources is provided relatively abundantly. Most resources have a network bandwidth 2-3x higher than the storage bandwidth.

Two-level Architecture. Based on the insight that these characteristics suggest effective resource combinations for cloud GNN training, we propose a two-level architecture utilizing relatively inexpensive GPU machines and high-performance memory machines with relatively large bandwidth, memory size, and CPU counts. The two-level architecture has the following advantages. First, users can selectively utilize the necessary resources. Unlike traditional DNN, GNN suffices with lower-performance GPUs but requires large I/O bandwidth and memory space, and the two-level architecture enables cluster configuration optimized for these requirements. Second, cloud resources can be used economically. Since single nodes with equivalent performance are significantly more expensive than cluster configurations, similar performance can be secured at much lower costs through a cluster configuration. Third, I/O bandwidth utilization suitable for cloud characteristics is possible. While cloud has limited storage bandwidth but sufficient network bandwidth, utilizing memory nodes can overcome storage bandwidth limitations with large network bandwidth.

We propose MEMORYBRIDGE, a system that cost-effectively accelerates GNN training in cloud environments based on this architecture. MEMORYBRIDGE utilizes affordable GPU resources as training nodes and compute resources without GPUs as memory nodes, instead of expensive single compute resources that include GPUs. MEMORYBRIDGE consists of two key modules. First, CLUSTERPLANNER is a mathematical solver that analyzes available resources through AMAT-based mathematical modeling to recommend the most cost-effective resource combination. Second, REMOTEGNN is a cloud-specialized GNN framework that minimizes I/O bottlenecks through fixed caching utilizing graph power-law and batch pipelining based on multiprocessing/multithreading. Specifically, memory machines efficiently manage frequently accessed graph data through degree-based caching strategy, and generated batches are asynchronously transferred to training machines utilizing the cloud's sufficient network bandwidth. Training machines maximize overall system throughput by overlapping GPU computation and data transfer.

Our contributions are as follows:

- We systematically analyzed three major challenges facing disk-based GNN in cloud environments: limitations of existing optimization techniques due to low I/O performance, inability to apply latest methodologies (e.g., inability to apply GPU-based I/O technologies), and difficulty in selecting optimal resource combinations.
- We proposed a cloud-specialized two-level architecture and MEMORYBRIDGE system as a comprehensive solution utilizing it. MEMORYBRIDGE accelerates GNN training in cloud environments through mathematical modeling-based resource optimization and efficient caching and pipelining considering graph characteristics.
- Through experiments, we demonstrated MEMORYBRIDGE's superiority in cost-efficiency, including cache hit rate and training time aspects compared to existing research.

We validated MEMORYBRIDGE by comparing it with the traditional GNN framework PyG [9] and the SOTA disk-based GNN methodologies Ginex [10] and MariusGNN [11]. The experimental results show that MEMORYBRIDGE achieved up to 32.7x faster training speed compared to existing methods, 9.9x cost efficiency compared to other cluster configuration methods, and achieved the same hit rate with 40% less caching size compared to other GNN caching systems.

II. BACKGROUND

A. GNN Structure and Training

GNN takes graph data as input. Graph data consists of graph structure data $G = (V, E)$ and feature vector data $H = \{ h_v, h_e \mid v \in V,\ e \in E \}$. Here, V is the node set, E is the edge set, and h_v, h_e are feature vectors for node v and edge e respectively. Depending on the type of graph data, at least one of h_v, h_e may not exist. The purpose of GNN is to generate appropriate embeddings for each node's "neighbors". Figure 1 shows the GNN training process.

GNN has multiple "layers" arranged in a vertical structure. The bottom layer "aggregates" and "operates" on neighboring

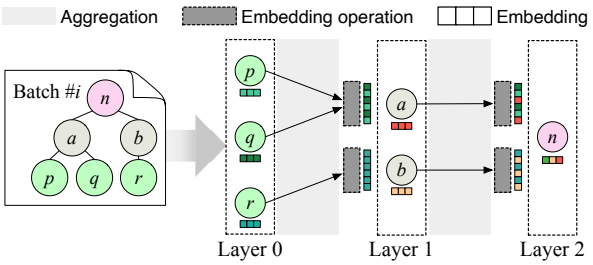


Fig. 1. GNN training process.

edge embeddings to generate embeddings, then passes the resulting embeddings to its upper layer. Upper layers collect and operate on embeddings from multiple lower layers to generate embeddings, and this process is recursively performed for all layers. Specifically, the output embedding h_v^n of the n-th layer for node v is calculated as in equation 1:

$$h_v^n = \phi\left(\left\{\, \alpha(h_u^{n-1}), \beta(h_{u\to v}) \;\middle|\; u \in \Gamma(v) \,\right\}\right) \tag{1}$$

Here, $\alpha(\cdot)$ is a function that aggregates embeddings of neighboring nodes and can be `max`, `sum`, or a more developed method, $\beta(\cdot)$ is a function that aggregates edges between v and its neighbors, $\phi(\cdot)$ is an embedding operation function, and $\Gamma(v)$ is the set of all neighboring nodes of v. According to this equation, each layer generates embeddings corresponding to first-order neighbors of a node, thus k layers are needed to generate k-th order embeddings of v.

Graph Neural Networks (GNNs) can perform full-batch training, where the entire graph is processed as a single input, or mini-batch training, where the graph is divided into smaller subgraphs for training. While full-batch training provides a holistic view of the graph, it becomes computationally prohibitive for large-scale graphs. Moreover, such graphs often make it challenging to capture localized information centered around specific nodes due to their size and complexity. To address these challenges, **subgraph sampling** has emerged as a promising approach, enabling the division of large graphs into manageable subgraphs for efficient learning. This paper specifically focuses on advancing mini-batch training methodologies for GNNs, leveraging subgraph sampling to balance computational efficiency and the preservation of critical graph structures. For this, equation 1 is performed by selecting N vertices called "seed nodes". One iteration produces N output embeddings, and the model is updated by calculating the loss value of the generated embeddings. In this mini-batch GNN training, the N edges and their k-th order neighbor data are called "batches" similar to traditional DNN, and N is called the batch size. However, k-hop neighbors for seed nodes can exist in very large numbers, increasing memory usage and training costs. To solve this, many subgraph sampling methods have been researched. For example, GraphSAGE limits the maximum number of edges that can be selected for each neighbor degree. Figure 2 shows GraphSAGE's sampling and batch generation for a given graph. From the first selected seed node, a maximum of 3 neighbors are selected, and a maximum

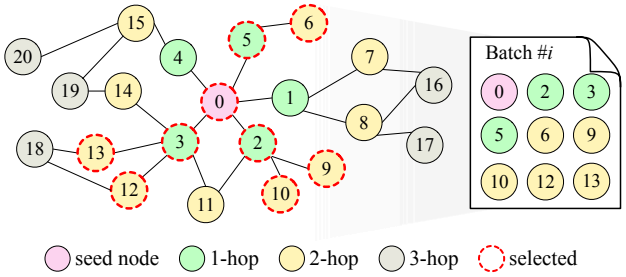


Fig. 2. Example of subgraph sampling proposed in GraphSAGE. Batch was generated by selecting 3 from 1-hop and 2 from 2-hop. Edges are omitted in the batch in the figure.

of 2 neighbors of those neighbors are selected to construct the subgraph. Such methods enable learning of large-scale graphs and utilization of local information of subgraphs.

Disk-based GNN Traditional GNN was performed by loading all graph data into memory, generating necessary batches to pass to GPU, and having GPU generate embeddings. However, graph data size has grown exponentially, recently exceeding the memory capacity of many single machines. To perform training in such environments, disk-based GNN was proposed, which uses disk space as memory extension instead of loading all data into memory. This enables training even on machines with relatively small memory by utilizing disk space instead of loading all data into memory. However, because storage must be accessed instead of memory to generate batches, batch generation speed became slower, and consequently, I/O bottleneck became a major factor in training speed degradation. Previous studies proposed many methods to solve this I/O bottleneck. Notable examples include feature vector caching [10], inter-device pipelining through data structure modification [11], and GPU-driven asynchronous I/O methods [12, 13].

B. Cloud Computing

Cloud computing is a service providing computing resources through the internet, and is being used for many workloads including DNN due to advantages such as flexibility, scalability, and ease of initial infrastructure setup. Cloud consists of three elements as follows:

- **Computing resources**. Called virtual machine (VM) instances, referring to resources including CPU, memory, and GPU. Users select from pre-defined resources instead of arbitrary combinations they desire.
- **Storage resources**. Corresponding to SSD, HDD, etc. in traditional computing, including not only block storage but also object storage, shared storage, etc.
- **Regions and networks**. Each resource belongs to specific network groups called 'regions', and each region and resource is connected by networks. Users pay fees according to resource usage time. Generally, higher-performance resources with more vCPUs or larger memory sizes require higher costs, and sometimes prices increase or decrease depending on demand.

TABLE I
COMPARISON OF NODE-LOCAL STORAGE AND AWS EBS STORAGE.

Type	Bandwidth		IOPS	
	Minimum	Maximum	Minimum	Maximum
Local NVMe	1,000	3,500	100K	750K
GP2	128	250	100	16,000
GP3	125	1,000	3,000	16,000

* AWS Northern Virginia, as of December 2024

TABLE II
AWS EBS PRICING POLICY

Type	Price per capacity	Price per IOPS	Price per bandwidth
GP2	$ 0.1 / GB	-	-
GP3	$ 0.08 / GB	$ 0.005 / IOPS	$ 0.04 / MB/s
io2	$ 0.125 / GB	$ 0.046 / IOPS	-

* AWS Northern Virginia, as of December 2024

Particularly, cloud storage resources have unique problems. Unlike local storage, especially node-local NVMe SSD, cloud storage has very slow I/O speed. This is because computing resources and storage resources are connected through networks rather than being physically connected. Table I shows theoretical bandwidths of typical node-local storage and cloud storage resources. GP2 is up to 14 times slower than node-local storage, and GP3 is 3.5 times slower. This indicates that even using maximum bandwidth will result in significantly slower I/O performance compared to node-local storage. Meanwhile, storage bandwidth in cloud is determined by the minimum of the storage resource's bandwidth and computing resource's bandwidth. For example, if using storage resources with 1000MB/s bandwidth and computing resources with 125MB/s storage bandwidth, actual storage bandwidth is limited to 125MB/s. In other words, computing resources and storage have very slow bandwidth compared to node-local even when using resources with the highest bandwidth, and if either has low bandwidth, the high bandwidth of the other becomes meaningless. Moreover, using high bandwidth requires paying much higher amounts compared to other storage solutions. Table II shows example prices according to storage capacity and bandwidth. When using 1TB GP3 storage at 1GB/s, $\frac{1}{3}$ of total storage cost must be paid for bandwidth cost. This makes storage I/O in cloud have very low cost efficiency compared to node-local.

C. Challenges of Disk-based GNN in Cloud Environments

Meanwhile, disk-based GNN has characteristics requiring very high storage I/O. This makes it very difficult for users to train disk-based GNN in cloud unlike traditional DNN. Unfortunately, previously mentioned existing research failed to consider these cloud characteristics. For Ginex [10], reading data for caching and creating and deleting files in storage instead becomes a major bottleneck in training. MariusGNN [11]'s storage-CPU memory-GPU pipelining is inefficient due to I/O speed degradation exceeding expectations. GIDS and Helios [12, 13] are inappropriate in cloud environments where only pre-defined instances can be used. Therefore, we must solve the following problem for GNN training in cloud: **Despite cloud resource characteristics, disk-based GNN training must be cost-effectively accelerated.**

III. TWO-LEVEL ARCHITECTURE

TABLE III
SPECIFICATIONS AND PRICES OF SOME AWS COMPUTE RESOURCES

Name	vCPU	Mem.	Strg B/W	Nwk B/W	Price
g4dn.8xlarge	32	128 GiB	9.5 Gbps	50 Gbps	$ 2.176
g4dn.4xlarge	16	64 GiB	4.75 Gbps	$\leq$ 25 Gbps	$ 1.204
g4dn.xlarge	4	16 GiB	64 GiB	$\leq$ 3.5 Gbps	$ 0.526
r8g.2xlarge	8	64 GiB	$\leq$ 10 Gbps	$\leq$ 15 Gbps	$ 0.471

* AWS Northern Virginia, as of December 2024

While cloud computing resources are broadly priced according to resource performance, detailed examination shows that pricing is not necessarily proportional to performance. In particular, resources including GPUs have higher prices than other resources due to demand. Table III shows prices and performance of some resources. We discovered the following interesting characteristics:

1) GPU resources using the same architecture provide identical GPU computational performance, with only values like memory size and bandwidth changing.
2) Resources without GPUs are provided at 39% of the price compared to resources with GPUs, despite offering similar levels of CPU counts, memory size, and storage bandwidth.
3) Storage bandwidth varies significantly with price. However, network bandwidth provides around 3.5Gbps even at lower prices.

This shows the possibility that combinations of inexpensive heterogeneous resources might be more cost-effective than single high-performance GPU resources. For example, instead of using the expensive GPU resource g4dn.8xlarge, a cluster can be created by combining the inexpensive GPU resource g4dn.xlarge with r8g.2xlarge, which has large memory capacity and high bandwidth despite not including a GPU. Specifically, using resource g4dn.xlarge as a GNN learning machine and resource r8g.2xlarge as a memory machine can provide a similar level of configuration to resource g4dn.8xlarge at a lower price. According to Table III, the combination of g4dn.xlarge and r8g.2xlarge results in up to 54% hourly cost savings compared to g4dn.8xlarge. Fortunately, this possibility of resource combination aligns well with GNN training characteristics. As observed earlier, while GPU resources are expensive, GNN has relatively low GPU computational requirements. This is because typical GNNs consist of 2-3 layers, requiring much less GPU computation compared to CNNs or Transformer models with dozens of layers. Additionally, using more layers reflects the entire graph, potentially disrupting local connectivity relationships. This is similar to how too large batch sizes in image processing can hinder learning. Therefore, disk-based GNN training has opportunities to utilize inexpensive GPU resources instead of expensive high-performance GPU resources. Instead, GNN requires large I/O bandwidth and memory space due to graph data characteristics and subgraph sampling. This naturally connects with another characteristic of cloud resources confirmed earlier, namely that high network bandwidth can be secured inexpensively even in resources without GPUs. As seen in observations (2) and (3) of

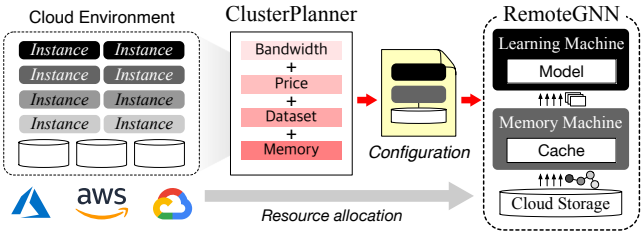


Fig. 3. Overall system overview of MEMORYBRIDGE. It uses a Two-level architecture divided into remote machine and learning machine, and consists of two modules, CLUSTERPLANNER and REMOTEGNN.

Table III, resources with high memory and network bandwidth without GPU can satisfy GNN's high I/O requirements while being relatively inexpensive.

IV. MEMORYBRIDGE DESIGN

A. Overview

MEMORYBRIDGE is a comprehensive solution enabling cost-effective GNN training in cloud environments. We use a two-level architecture consisting of two machines: a memory machine that reads and processes graph data to generate batches, and a learning machine that performs actual training. Under this architecture, MEMORYBRIDGE recommends the most cost-effective cluster and enables efficient GNN training. For this, MEMORYBRIDGE consists of two modules: CLUSTERPLANNER and REMOTEGNN.

- CLUSTERPLANNER is a mathematical solver that recommends resources expected to be most cost-effective based on available resources to configure clusters. CLUSTERPLANNER operates before users allocate resources, enabling economical cloud utilization.
- REMOTEGNN is a GNN training framework that optimizes I/O during GNN training in two-level architecture clusters, thereby cost-effectively accelerating training. REMOTEGNN accelerates GNN training in cloud clusters configured with two-level architecture.

Figure 3 shows the architecture proposed by MEMORYBRIDGE and MEMORYBRIDGE's design overview. Specifically, the process of performing GNN training using MEMORYBRIDGE in cloud environments is as follows: (1) Before resource allocation, CLUSTERPLANNER recommends efficient resource combinations based on user's available resources. (2) Users allocate cloud computing and storage resources based on CLUSTERPLANNER's recommendations. (3) On allocated resources, GNN training is performed by writing and executing GNN training code using REMOTEGNN. This process is designed to be very easy for users wanting to perform training using GNN frameworks in cloud environments. In the following subsections, we describe each module's design objectives and specific operation methods.

B. CLUSTERPLANNER Design

Goal. While two-level architecture has the advantage of utilizing cloud resources efficiently at affordable prices, incorrect clustering can actually degrade performance. Therefore, we aim to find the most cost-effective architecture by considering both performance and price simultaneously. This requires considering I/O bandwidth along with computing resource prices. I/O bandwidth can affect time, and time can affect cost. Too small memory can affect caching performance increasing training time, while too large memory resources may require paying high costs. CLUSTERPLANNER enables cost-effective clustering through mathematical modeling that comprehensively considers these factors.

TABLE IV
CLUSTERPLANNER MODELING NOTATION

Notation	Description
$\|G\|$	Graph dataset size (GB)
$\mathbf{I}$	Complete instance set
$\mathbf{I}_{GPU}$	Set of GPU instances
v_k	k-th instance vector
$\mathcal{M}_k$	Memory size of k-th instance (GB)
β_k^s	Storage bandwidth of k-th instance (MB/s)
β_k^n	Network bandwidth of k-th instance (Gbps)
p_k	Hourly cost of k-th instance (USD/hour)
β_x	Bandwidth of storage resource x (MB/s)
$\omega(\beta_x)$	Bandwidth price of storage resource x with bandwidth β_x (USD/hour)
$\phi(s,x)$	Inverse of effective bandwidth (s/MB). Calculated as $\max\left(\frac{1}{s}, \frac{1}{x}\right)$

Notation. We define basic elements for modeling as shown in Table IV: Here, $\mathbf{I} = \mathbf{I}_{GPU} \cup \overline{\mathbf{I}_{GPU}}$ (where $\overline{A}$ is the complement set of A), and each instance vector is defined as $v_k = (\mathcal{M}_k, \beta_k^s, \beta_k^n, p_k)$.

Assumptions. Disk-based GNN training consists of the following systematic steps: loading data from storage $(\mathcal{A})$ - generating batch in CPU $(\mathcal{B})$ - transferring data to GPU memory $(\mathcal{D})$ - performing training in GPU $(\mathcal{E})$. For two-level architecture, network transfer between memory machine and learning machine $(\mathcal{C})$ is added. For system performance modeling, we establish assumptions based on the following observations:

- Batch generation in CPU $(\mathcal{B})$ is relatively negligible compared to storage and network I/O time as it accesses local memory.
- GPU memory transfer $(\mathcal{D})$ is transfer through internal bus line, with minimal differences between architectures and computing resources.
- Since GNN's main bottleneck is in memory access, GPU computation time $(\mathcal{E})$ has limited impact on overall performance, and simultaneously, differences between GPU computing resources are minimal for GNN model training.

Therefore, total execution time for single-machine structure can be expressed as $T_1 = T_1(\mathcal{A})$, and total execution time for two-level architecture as $T_2 = T_2(\mathcal{A}, \mathcal{C})$. However, to accurately model these I/O times, following system characteristics must be considered: (1) storage I/O is simultaneously affected by storage's own bandwidth and system's storage bandwidth, (2) data access occurs with specific probability rather than always occurring, and (3) in two-level architecture, data access patterns change due to presence of memory machine. To systematically model these complex I/O characteristics, we apply the AMAT(Average Memory Access Time) concept:

$$T = h_t + m_r \cdot m_p \tag{2}$$

where h_t is memory access time, m_r is memory miss rate, and m_p is miss penalty. Based on this, we modeled time, performance, and price when training disk-based GNN according to architecture as follows.

Single Machine Modeling. A GPU computing resource is selected from $\mathbf{I}_{GPU}$ for training execution. At this time, m_p is affected by storage latency. However, latency variation occurs depending on the network state of the region in cloud and the actual physical server configuration method, making it difficult to know accurate latency at CLUSTERPLANNER's prediction point. Fortunately, according to our observation shown in Figure 5, latency and storage bandwidth have a close correlation. Therefore, we use storage bandwidth β_k^s as a substitute value for latency. Meanwhile, h_t is negligibly small compared to I/O time determined by storage bandwidth, and since all data is stored in storage in basic disk-based GNN, m_r is 1. Therefore, when storage resource bandwidth is β_x, time T_1, hourly price P_1, and cost C_1 are obtained as in equation IV-B:

$$T_1(v_k, \beta_x) = |G| \times \phi(\beta_k^s, \beta_x) \tag{3}$$

$$P_1(v_k, \beta_x) = \omega(\beta_x) + p_k \tag{4}$$

$$C_1(v_k, \beta_x) = T_1 P_1(v_k, \beta_x) \tag{5}$$

where $\phi(s, x) = \max\left(\frac{1}{s}, \frac{1}{x}\right)$ is the inverse of effective bandwidth, representing data transfer time limited by lower bandwidth between computing resource and storage resource, and $\omega(\beta^s)$ is storage cost per bandwidth, multiplied by inverse of ϕ to represent storage resource cost.

Two-level Cluster Modeling. Following the two-level architecture explained in Section IV, select one instance each from $\mathbf{I}_{GPU}$ and $\overline{\mathbf{I}_{GPU}}$: computing machine $v_k \in \mathbf{I}_{GPU}$ and memory machine $v_l \in \overline{\mathbf{I}_{GPU}}$. Total bandwidth is determined by network bandwidth of each machine and storage bandwidth of memory machine. At this time, since β^n is sufficiently larger than storage bandwidth β^s, we can assume bandwidth is limited by storage bandwidth, and h_t value in equation 2 is negligibly small. Since batch generation task accessing storage is performed on memory machine, m_r in equation 2 is the ratio cached in memory machine, and m_p is disk reading speed in memory machine. Therefore, cost C_2 can be obtained as follows:

$$T_2(v_k, v_l, \beta_x) = \left(1 - \frac{\mathcal{M}_l}{|G|}\right) \times |G| \times \phi(\beta_l^s, \beta_x) \tag{6}$$

$$P_2(v_k, v_l, \beta_x) = \omega(\beta_x) + p_k + p_l \tag{7}$$

$$C_2(v_k, v_l, \beta_x) = T_2 P_2(v_k, v_l, \beta_x) \tag{8}$$

The obtained C_1, C_2 are values considering I/O time and computing resource costs, representing cost-effectiveness of cloud for performing GNN workloads where I/O time accounts for most of total time. CLUSTERPLANNER receives list of available resources, calculates this to obtain $|\mathbf{I}_{GPU}|$ tensors $\mathbf{C}_1$ and $|\mathbf{I}_{GPU}| \times \left|\overline{\mathbf{I}_{GPU}}\right|$ tensors $\mathbf{C}_2$. By comparing and analyzing these cost tensors, it identifies a more cost-effective configuration between single machine and two-level architecture, and recommends optimized instance combination for that configuration to users. In our observation, CLUSTERPLANNER always suggests two-level structure instead of single machine. Detailed analysis is provided in Section VII-B.

C. Design of REMOTEGNN

Figure 4 depicts REMOTEGNN's operation process. REMOTEGNN reduces I/O bottlenecks and accelerates overall training by quickly generating batches through graph data caching in memory machine and transferring generated batches to learning machine through network I/O, which is relatively faster than storage I/O. The training process is as follows. During initialization, the memory machine analyzes graph information and caches necessary data. This is a one-time process and recaching does not occur until training ends. When training begins, the memory machine explores the graph according to given model to generate batches, and generated batches undergo serialization before being transferred to learning machine through internal network communication. Batch generation process and batch transfer network communication overlap to minimize I/O stalls. The learning machine deserializes received batches and transfers them to GPU. Batches are transferred to batch queue, and the loader finds data from batch queue for training. Each data reception, storage, and training is parallelized like memory machine to minimize stalls as much as possible.

Batch Pipelining. REMOTEGNN performs multiprocessing- and multithreading-based batch parallelization to minimize I/O stalls caused by unnecessary synchronization operations during GNN sampling while ensuring batches are generated and transferred at sufficient speed. Before epoch starts, the sequence creator in remote machine generates batch sequences, which are sets of seed node bundles. Multiple batch creator processes sequentially read batch sequences and explore graph structure from seed nodes to generate blueprint batches. Blueprint batches contain metadata about graph structure and metadata of data requiring I/O, such as feature values. Processes that generate blueprint batches insert them into vector queue. At this time, vector queue is shared memory space, and since multiple processes simultaneously insert data, the order in which blueprint batches are inserted into queue may vary when context switching occurs. However, considering the grammar of existing graph learning that dynamically generates batches during training, this is not problematic. Vector loader processes consume vector queue and make following decisions: (1) If data is not in memory, read data from storage. Data is loaded in memory-mapped (mmap) format. (2) If data is already in memory or cache, use that data. If all data is in memory or cache, vector loaders serialize each data to turn blueprint batches into actual batches. These batches are passed to batch sender processes, and batch sender processes transfer them to learning machine through network communication. In this process, remote machine's 4 modules interact with each other but do not operate synchronously, thus enabling I/O-effective data processing. In learning machine, batch receiver thread receives batch data. Received batches are deserialized

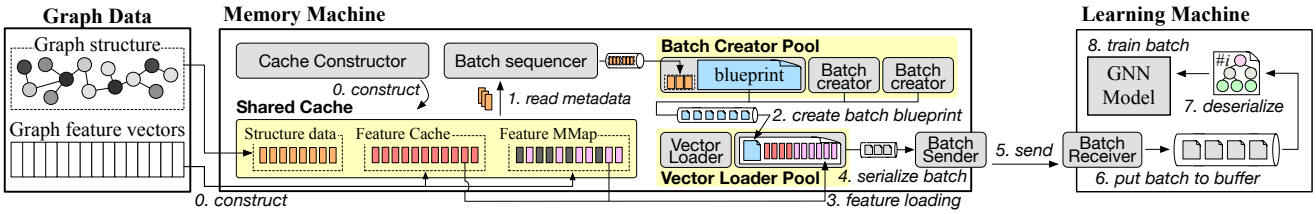
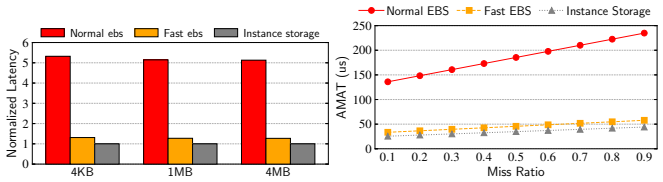


Fig. 4. GNN training mechanism using REMOTEGNN.



(a) Read Latency (b) AMAT

Fig. 5. Analysis of cloud storage resources for predicting cache system effectiveness. (a) Read testing with 4KB, 1MB, 4MB units on various storage resources (b) AMAT analysis according to cache miss rate of 4KB data

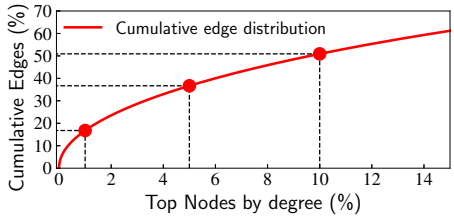


Fig. 6. Cumulative edge distribution of OGBN-Papers100M dataset. x-axis represents ratio from nodes with highest degree, y-axis represents cumulative ratio of total edges included in those nodes.

and transferred to GPU for use in training, and model training using those batches is performed on GPU.

V. CACHE SYSTEM DESIGN

In existing disk-based GNN research, caching primarily focused on predicting data needed for batches and caching it. Ginex [10], one of the representative caching systems, calculated feature vectors to be accessed in advance to enable processes to efficiently access feature vector data during batch generation. However, in cloud environments, limitations of dynamic caching methods become prominent due to very high storage access latency. Methods that predict and dynamically cache data generate multiple disk I/Os, causing severe performance degradation in cloud environments. We compared three types: 125MB/s EBS (Normal EBS), 500MB/s (Fast EBS), and EBS physically attached to compute resource (Instance storage). Figure 5(a) compares latency when reading 4KB, 1MB, 4MB data from storage. Each time is normalized based on the fastest Instance storage latency. Experimental results showed similar latency patterns by bandwidth for EBS. Particularly, commonly used EBS showed over 5 times longer latency compared to physically attached EBS. This means it could be quite critical to overall performance considering GNN's work pattern requiring reading small data. Meanwhile, frequent cache replacement under high read latency can paradoxically mask cache effectiveness. Figure 5(b) shows AMAT according to cache miss rate calculated based on read latency shown in Figure 5(a). Compared to the other two EBS types, normal EBS has significantly higher penalty for cache misses. For example, for a miss rate around 0.5, AMAT is up to 6 times higher. Through this, we can infer that high cache miss rates will have substantial impact on overall training performance. For effective caching in such environments, we observed following locality in graph data: First, graph structure data has high locality as it is accessed every batch because neighbor nodes must be identified for subgraph sampling. Second, degrees of graph nodes are not uniform. Instead, they follow power law where certain nodes monopolize many edges. Figure 6 shows cumulative edge distribution according to number of nodes when nodes of the papers100M dataset are sorted in descending order by degree. Nodes in the top 1% by degree are connected to 13.2% of total edges, top 5% to 31.3%, and top 10% to 44.5%. Therefore, nodes with high degrees have much higher probability of being accessed during batch generation compared to other nodes, thus having high locality.

Graph-Aware Fixed Caching. Utilizing characteristics observed above, REMOTEGNN caches graph structure data and feature vector data of high-degree nodes for efficient caching. Graph structure data is stored in CSC(Compressed Sparse Column) and CSR(Compressed Sparse Row) format enabling both external and internal edge exploration in $O(1)$ time. This requires paying more memory space. For papers100M, graph structure data occupies about 31.3% of total memory, but this is an efficient trade-off considering performance gain from reducing repetitive cloud storage access. For feature vector data of high-degree nodes, feature vectors of nodes with higher degrees are cached preferentially. All available memory space excluding essential data and graph structure data is utilized for feature vector caching. Detailed analysis of cache performance is provided in Section VII-C.

VI. IMPLEMENTATION

We implemented MEMORYBRIDGE using Python (3.10.12). Specifically, CLUSTERPLANNER for cloud architecture provisioning used SciPy (1.11.2), one of Python libraries. REMOTEGNN was developed in approximately 1.2K lines of Python code using PyG [9] (2.6.1), one of the latest GNN training frameworks. Specifically, batch generation in memory machine operates identically to PyG's `Neighborloader`. For inter-machine communication, we

TABLE V
GRAPH DATASETS

Type	Name	Nodes.	Edges.
Large	Papers100M	111M	3.3B
	MAG240M	244M	3.4B
	Friendster	66M	3.6B
Small	Arxiv	169K	1M

used OpenMPI (v4.1.4), one of the MPI implementations, and defined new communication functions in approximately 200 lines of C++ code for MPI transfer optimization. For data processing, we used Numpy (1.25.1) from Python libraries, and for GPU training, we used PyTorch (2.5.1). Users wanting to use REMOTEGNN for training can perform training using `DistributedGraphLoader` instead of PyG's `Neighborloader`. Users can perform GNN training using identical code in both learning machine and memory machine, and users familiar with PyG training can easily use REMOTEGNN with existing code. This is possible because REMOTEGNN provides abstraction of all caching and data transfer.

VII. EVALUATION

A. Methodology

Environment. We conducted experiments in Amazon Web Service (AWS) cloud environment. The experiments were performed between September and December 2024, using the Northern Virginia region (us-east-1). We considered G4dn, P2, P3 GPU instances, and for memory resources, we considered R6, R7, R8, C7, and C8 instances. We limited instance memory to 64GB to simulate scenarios where graph data exceeds single machine memory. For storage, we used GP3 general-purpose SSD with 2.5TB capacity, default bandwidth of 125MB/s. IOPS was also set to the cloud default of 3,000. This reflects assumptions about typical cloud environments used by users.

GNN Model. We selected GraphSAGE [14] as the GNN model for learning graph data. GraphSAGE can perform training by generating subgraphs, avoiding the need to learn all data at once. GraphSAGE is currently the most widely used method and is appropriate for testing our system that performs optimization during training through subgraph sampling.

Graph Datasets. For large-scale graph data experiments, we used OGBN-Papers100M, OGB-MAG240M, and Friendster. All of these exceeded single machine memory capacity after preprocessing, while simultaneously being learnable by the comparison targets to be discussed. Meanwhile, to validate REMOTEGNN in in-memory training environments where datasets are sufficiently smaller than memory, we used the small-scale dataset OGBN-ARXIV. Detailed information about the datasets is shown in Table V.

Baselines. We compare our solution with the following three frameworks:

- PyG [9] with `mmap()`
- Ginex [10]
- MariusGNN [11]

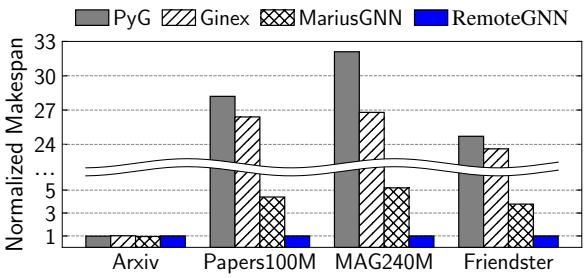


Fig. 7. Comparison of training speed by dataset for GNN training frameworks. All values are normalized based on REMOTEGNN's execution time for each dataset.

Since disk-based GNN training is difficult to work well in PyG, we chose the most basic method using mmap. We could not compare our solution with Helios [12] as it is not open source. GIDS [13] was excluded as it uses GPU-driven asynchronous I/O which cannot be applied in cloud. Explanations for Ginex and MariusGNN are provided in sections II-A and II-B.

B. End-to-end Training Performance

Figure 7 compares total end-to-end time with both preprocessing and training phases among four GNN frameworks.All times are normalized to REMOTEGNN's execution time. PyG and Ginex showed up to 32.7 times and 26 times slower performance compared to REMOTEGNN. PyG's continuous use of mmap results in slow disk I/O performance directly affecting training. For Ginex, the process of creating 'neighbor cache' involves considerable disk I/O, resulting in very slow performance. This is a key difference from our caching system. While MariusGNN is faster than the other two methods, it remains slow. This is because MariusGNN involves significant I/O when moving data between disk, CPU memory, and GPU, repeatedly accessing disk in the process, causing the entire pipelining to stall. In contrast, our solution showed minimum time across all large-scale datasets. This can be interpreted in two ways. First, disk I/O was minimized through caching. This minimizes disk access by internal modules, enabling effective pipelining. Second, the structure recommended by CLUSTERPLANNER operates efficiently. Meanwhile, OGBN-Arxiv is a small dataset that can fit all data in memory. In this case, REMOTEGNN's training time was about 1-2% longer compared to other solutions. This means network communication overhead is minimal, indicating the validity of storage speed-centered approach.

REMOTEGNN generates batches randomly on remote machines that match the runtime environment rather than directly on training machines. To verify the integrity of this batch generation and transfer process, we compared the validation accuracy of a GraphSAGE model trained on the Papers100M dataset using both PyG and REMOTEGNN at each epoch, as shown in Figure 8. Since batches are randomly generated at runtime, batch composition and order are completely different between the two methods at each epoch, resulting in different accuracies. However, these values are negligible in overall

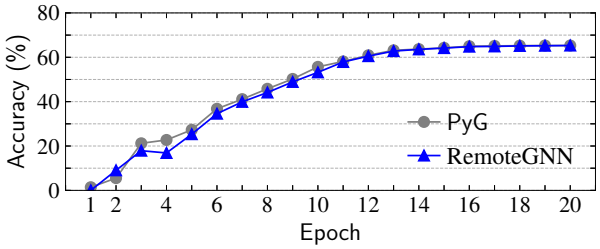


Fig. 8. Comparison of learning accuracy between PyG and REMOTEGNN. Shows accuracy at each epoch when training GNN model using Papers100M dataset for 20 epochs.

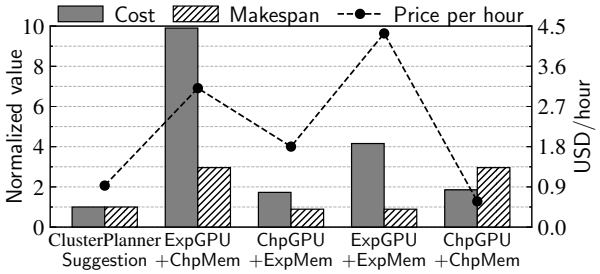


Fig. 9. Normalized cost and makespan for different machine combinations ('Exp': expensive, 'Chp': cheap), including CLUSTERPLANNER suggestion. The black dashed line (−−) represents the price per hour.

trends, and two different training frameworks using the same model converge with similar accuracy patterns. This demonstrates that REMOTEGNN's batch generation method, network transfer, and inter-process pipelining neither manipulate batch data nor affect model accuracy.

C. Analysis of Cost-effectiveness

We validated how economically CLUSTERPLANNER recommends clusters. To our knowledge, MEMORYBRIDGE is the first solution considering two-level clusters from disk-based GNN perspective, and CLUSTERPLANNER is the first solver modeling this. Therefore, we compared CLUSTERPLANNER with four arbitrary configuration methods. Each configuration method selects one GPU and one Non-GPU machine, choosing the cheapest and highest-performing machines. Figure 9 shows actual costs and makespan when performing GNN training using REMOTEGNN after allocating actual resources with each configuration method. Each cost and makespan is normalized to CLUSTERPLANNER's suggested method. Experimental results showed CLUSTERPLANNER's recommendation was lowest in both cost and makespan aspects, providing up to 9.9 times better cost-efficiency compared to other configurations such as 'ExpGPU + ChpMem'. Despite higher hourly rates ($0.93 vs. $0.576), CLUSTERPLANNER's recommendation achieved 2.8 times lower total cost than 'ChpGPU + ChpMem' due to significantly reduced makespan. This means CLUSTERPLANNER recommends cost-effective combinations considering performance instead of simply recommending combinations with economical hourly prices. Simultaneously, it showed up to 12% longer makespan than using 'ExpGPU+ExpMem' combination. This results from I/O performance improvement due to vCPU differences and cache

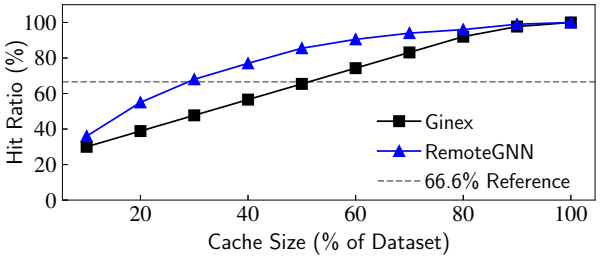


Fig. 10. Comparison of caching systems between Ginex(black) and REMOTEGNN(blue). Shows hit rate versus cache size relative to total dataset. Green line indicates 66.6% hit rate.

effects due to memory size differences. However, total cost showed 4-fold difference. This indicates CLUSTERPLANNER appropriately reflects trade-off relationship between cost and time to recommend cost-effective combinations.

D. Analysis of Caching System

To validate our approach, we compared REMOTEGNN's caching system with Ginex's disk-based GNN training cache implementation, with Figure 10 illustrating the resulting cache hit rates measured across various cache sizes. Ginex shows linear increase trend. This is because it guarantees caching of certain number of feature vectors according to Belady algorithm. In this case, about 50% of data must be cached to achieve 66.6% hit rate. However, this method doesn't work effectively in cases with limited memory like cloud. When learning with memory that can cache about 30% of total dataset, hit rate falls short of 50%. Meanwhile, REMOTEGNN prioritizes caching features of nodes with high degrees among all features. This results in better cache hit rates. Similarly, to achieve 66.6% hit rate, REMOTEGNN only needs to cache about 30% of the dataset. This indicates REMOTEGNN's caching system is operating effectively.

VIII. RELATED WORK

Graph Processing Systems. Even before GNN became promising, system research for processing large-scale graph data [15, 16, 17, 18] was actively conducted. GraphChi [15] enabled large-scale graph processing on a single personal computer (PC) through parallel sliding window method. X-stream [16] minimized disk access through edge-centric approach. FlashGraph [17] and MOSAIC [18] improved graph processing performance by introducing SSD-optimized data structures.

GNN Training Framework Systems. Efficient learning of large-scale graph data emerged as a major challenge as GNN became an established deep learning model, prompting various system solutions. Distributed system methods [19, 20, 21] propose utilizing resources of multiple machines for large-scale graph processing. ROC [19] proposed distributed multi-GPU framework applying memory management and graph partitioning. NeuGraph [20] successfully processed large-scale graphs that couldn't be processed on an existing single GPU. DistDGL [21] extended existing DGL library to distributed systems. Disk-based GNN frameworks proposed performance

improvement techniques, including Belady algorithm-based caching [10], optimized data structures and pipeline optimization [11], and GPU-initiated asynchronous I/O [12, 13].

Caching Systems. Several caching strategies [10, 22, 23, 24, 25] were proposed to prevent repeated I/O operations, enhancing data movement and processing efficiency for improved GNN training performance. GRASP [22] proposed caching based on node degree, while Graphfire [23] managed cache by learning access patterns. P-OPT [24] implemented near-optimal caching using adjacency matrix transposition, PaGraph [25] utilized GPU resources in single-server multi-GPU environments, and Ginex [10] developed more advanced caching systems. However, these methods couldn't achieve performance equal to their target architectures as they didn't consider cloud characteristics.

Resource Optimization Research. Resource optimization research [26, 27, 28, 29, 30] explored cost-effective cloud environment usage. H. Wang et al. [26] and S. Deochake [27] analyzed cloud pricing systems and cost reduction methods. P. Kokkinos et al. [28] proposed utilization and cost-based cluster configuration. E. M. Malta et al. [29] presented methodology for selecting cost-effective instances for DL applications. DeepVM [30] proposed a practical clustering using Spot VMs for image tasks. However, these studies have limitations in not achieving practical clustering or specifically verifying other tasks including GNN.

IX. Conclusion

We proposed MemoryBridge, which can effectively accelerate GNN training in cloud environments. MemoryBridge recommends cost-effective cluster configurations considering various cloud characteristics, predefined support, slow storage I/O speeds, price efficiency, and enables training on these clusters. To our knowledge, MemoryBridge is the first solution considering GNN in cloud environments. MemoryBridge achieved the maximum cost efficiency compared to existing methods and effectively accelerated training.

Acknowledgement

This work was partly supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2022-0-00498, Development of high-efficiency AI computing SW core technology for high-speed processing of large learning models) and the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(RS-2025-00564249 and RS-2024-00416666).

References

[1] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," *Journal of Machine Learning Research*, vol. 23, no. 89, pp. 1–64, 2022.

[2] A. Strokach, D. Becerra, C. Corbi-Verge, A. Perez-Riba, and P. M. Kim, "Fast and flexible protein design using deep graph neural networks," *Cell systems*, vol. 11, no. 4, pp. 402–411, 2020.

[3] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, "Graph neural networks for social recommendation," in *The world wide web conference*, WWW'19, pp. 417–426, 2019.

[4] A. Pal, C. Eksombatchai, Y. Zhou, B. Zhao, C. Rosenberg, and J. Leskovec, "Pinnersage: Multi-modal user embedding framework for recommendations at pinterest," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD'20, pp. 2311–2320, 2020.

[5] 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*, KDD'18, pp. 974–983, 2018.

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

[7] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[8] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, WWW'15, pp. 1067–1077, 2015.

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

[10] Y. Park, S. Min, and J. W. Lee, "Ginex: SSD-enabled billion-scale graph neural network training on a single machine via provably optimal in-memory caching," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 2626–2639, 2022.

[11] R. Waleffe, J. Mohoney, T. Rekatsinas, and S. Venkataraman, "Mariusgnn: Resource-efficient out-of-core training of graph neural networks," in *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys'20, pp. 144–161, 2023.

[12] J. Sun, M. Sun, Z. Zhang, J. Xie, Z. Shi, Z. Yang, J. Zhang, F. Wu, and Z. Wang, "Helios: An Efficient Out-of-core GNN Training System on Terabyte-scale Graphs with In-memory Performance," *arXiv preprint arXiv:2310.00837*, 2023.

[13] J. B. Park, V. S. Mailthody, Z. Qureshi, and W.-m. Hwu, "Accelerating sampling and aggregation operations in gnn frameworks with gpu initiated direct storage accesses," *arXiv preprint arXiv:2306.16384*, 2023.

[14] J. Liu, G. P. Ong, and X. Chen, "GraphSAGE-based traffic speed forecasting for segment network with sparse data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 1755–1766, 2020.

[15] A. Kyrola, G. Blelloch, and C. Guestrin, "GraphChi:Large-Scale graph computation on just a PC," in *10th USENIX symposium on operating systems design and implementation*, OSDI'12, pp. 31–46, 2012.

[16] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-centric graph processing using streaming partitions," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 472–488, 2013.

[17] D. Zheng, D. Mhembere, R. Burns, J. Vogelstein, C. E. Priebe, and A. S. Szalay, "FlashGraph: Processing Billion-Node graphs on an array of commodity SSDs," in *13th USENIX Conference on File and Storage Technologies*, FAST'15, pp. 45–58, 2015.

[18] S. Maass, C. Min, S. Kashyap, W. Kang, M. Kumar, and T. Kim, "Mosaic: Processing a trillion-edge graph on a single machine," in *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys'17, pp. 527–543, 2017.

[19] 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*, vol. 2, pp. 187–198, 2020.

[20] 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*, ATC'19, pp. 443–458, 2019.

[21] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, "DistDGL: Distributed graph neural network training for billion-scale graphs," in *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms*, IA3'20, pp. 36–44, IEEE, 2020.

[22] P. Faldu, J. Diamond, and B. Grot, "Domain-specialized cache management for graph analytics," in *2020 IEEE International Symposium on High Performance Computer Architecture*, HPCA'20, pp. 234–248, IEEE, 2020.

[23] A. Manocha, J. L. Aragón, and M. Martonosi, "Graphfire: Synergizing fetch, insertion, and replacement policies for graph analytics," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 291–304, 2022.

[24] V. Balaji, N. Crago, A. Jaleel, and B. Lucia, "P-opt: Practical optimal cache replacement for graph analytics," in *2021 IEEE International Symposium on High-Performance Computer Architecture*, HPCA'21, pp. 668–681, IEEE, 2021.

[25] Z. Lin, C. Li, Y. Miao, Y. Liu, and Y. Xu, "Pagraph: Scaling gnn training on large graphs via computation-aware caching," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC'20, pp. 401–415, 2020.

[26] H. Wang, Q. Jing, R. Chen, B. He, Z. Qian, and L. Zhou, "Distributed systems meet economics: pricing in the cloud," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing*, HotCloud'10, 2010.

[27] S. Deochake, "Cloud Cost Optimization: A Comprehensive Review of Strategies and Case Studies," 2023.

[28] P. Kokkinos, T. A. Varvarigou, A. Kretsis, P. Soumplis, and E. A. Varvarigos, "Cost and utilization optimization of amazon ec2 instances," in *Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing*, pp. 518–525, IEEE, 2013.

[29] E. M. Malta, S. Avila, and E. Borin, "Exploring the cost-benefit of aws ec2 gpu instances for deep learning applications," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, pp. 21–29, 2019.

[30] Y. Kim, K. Kim, Y. Cho, J. Kim, A. Khan, K.-D. Kang, B.-S. An, M.-H. Cha, H.-Y. Kim, and Y. Kim, "DeepVM: Integrating Spot and On-Demand VMs for Cost-Efficient Deep Learning Clusters in the Cloud," in *2024 IEEE 24th International Symposium on Cluster, Cloud and Internet Computing*, CCGRID'24, pp. 227–235, 2024.