GraNNDis: Fast Distributed Graph Neural Network Training Framework for Multi-Server Clusters

Jaeyong Song
Seoul National University
Seoul, South Korea
jaeyong.song@snu.ac.kr

Hongsun Jang
Seoul National University
Seoul, South Korea
hongsun.jang@snu.ac.kr

Hunseong Lim
Seoul National University
Seoul, South Korea
hunseong.lim@snu.ac.kr

Jaewon Jung
Seoul National University
Seoul, South Korea
jungjaewon@snu.ac.kr

Youngsok Kim
Yonsei University
Seoul, South Korea
youngsok@yonsei.ac.kr

Jinho Lee*
Seoul National University
Seoul, South Korea
leejinho@snu.ac.kr

ABSTRACT

Graph neural networks (GNNs) are one of the rapidly growing fields within deep learning. While many distributed GNN training frameworks have been proposed to increase the training throughput, they face three limitations when applied to multi-server clusters. 1) They suffer from an inter-server communication bottleneck because they do not consider the inter-/intra-server bandwidth gap, a representative characteristic of multi-server clusters. 2) Redundant memory usage and computation hinder the scalability of the distributed frameworks. 3) Sampling methods, de facto standard in mini-batch training, incur unnecessary errors in multi-server clusters.

We found that these limitations can be addressed by exploiting the characteristics of multi-server clusters. Here, we propose *GraNNDis*, a fast distributed GNN training framework for multi-server clusters. Firstly, we present *Flexible Preloading*, which preloads the essential vertex dependencies server-wise to reduce the low-bandwidth inter-server communications. Secondly, we introduce *Cooperative Batching*, which enables memory-efficient, less redundant mini-batch training by utilizing high-bandwidth intra-server communications. Thirdly, we propose *Expansion-aware Sampling*, a cluster-aware sampling method, which samples the edges that affect the system speedup. As sampling the intra-server dependencies does not contribute much to the speedup as they are communicated through fast intra-server links, it only targets a server boundary to be sampled. Lastly, we introduce *One-Hop Graph Masking*, a computation and communication structure to realize the above methods in multi-server environments. We evaluated GraNNDis on multi-server clusters, and it provided significant speedup over the state-of-the-art distributed GNN training frameworks. GraNNDis is open-sourced at https://github.com/AIS-SNU/GraNNDis_Artifact to facilitate its use.

---

*Corresponding author.

CCS CONCEPTS

• **Computing methodologies → Distributed computing methodologies**; **Artificial intelligence**; • **Computer systems organization → Distributed architectures**.

KEYWORDS

Distributed Systems, Systems for Deep Learning, Graph Neural Network Training, Distributed Graph Neural Network Training

1 INTRODUCTION

Over the last few years, graph neural networks (GNNs) have attracted increasing attention among various deep learning tasks. As graphs can represent many types of non-structured data that do not fit into structured formats (e.g., images, or texts), they are widely used in various fields, such as social network user clustering [15], protein analysis [18], physics [63], and even traditional machine learning problems [8, 27]. Since the initial establishment of a few pioneering GNN algorithms [3, 10, 34], various algorithms have been proposed [23, 74, 82, 93] that capture different aspects of the knowledge within graph data. The development of these algorithms was also accompanied by advancements in the training frameworks that provide higher throughput and convenient interfaces for implementation [16, 28, 78].

To achieve a higher training throughput, many distributed GNN training frameworks [19, 28, 45–47, 80, 94, 96, 97, 99] have been proposed. Full-batch-based frameworks [28, 56, 76, 77] try to reduce the communication bottleneck of distributed training. Mini-batch-based frameworks [30, 31, 96, 97] mitigate the redundant computation and memory usage of mini-batch training from the neighbor explosion through sampling approaches [6, 9, 23, 58, 60, 93].

However, when conducting distributed GNN training on multi-server clusters, we found several challenges that must be addressed:

(1) *Inter-Server Communication Bottleneck.* While some researches [56, 76, 77] address the communication bottleneck of distributed full-batch GNN training, they only focused on multi-GPU cases.

As even high-speed inter-server links (around a few tens of GB/s) [50] are two orders of magnitude lower than the GPU device memory bandwidth [29], and several times slower than intra-server links [38] in multi-server clusters, inter-server communication becomes the new bottleneck.

(2) *Redundant Memory Usage/Computation.* Multi-server clusters are adopted for GNN training, especially when using large graphs [26, 33] or deep models [39–41, 49]. With such cases, redundant memory usage and computation still exist even when sampling methods [6, 9, 23, 58, 60, 92, 93] are applied.

(3) *Unnecessary Error from Sampling.* Existing sampling approaches [6, 23, 92, 93] are manipulating the whole graph structure. However, in multi-server clusters, sampling the intra-server edges (dependencies) does not contribute much to increasing throughput, as they are communicated through the fast intra-server links. Therefore, we need another sampling strategy to reduce the unnecessary information loss from a sampling.

Fortunately, we identified that these challenges can be addressed by utilizing the opportunity provided by the characteristics of multi-server clusters. Here, we propose *GraNNDis*[1], a fast distributed graph neural network training framework for multi-server clusters.

Firstly, we propose flexible preloading, in which each server preloads the vertex dependencies required and distributes them to the intra-server GPUs. This strategy utilizes intra-server links instead of inter-server ones, thus reducing the inter-server communication bottleneck. Since it requires additional memory, flexible preloading dynamically selects the preloading hyperparameter considering the remaining server-wise aggregated GPU memory.

Secondly, we present cooperative batching, which creates large-sized batches for mini-batch training by merging the GPU-wise mini-batches of existing frameworks. While previous mini-batch methods suffer from a significant redundancy issue, cooperative batching addresses it by aggregating multiple GPUs in a server with fast intra-server links.

Thirdly, we suggest expansion-aware sampling, a new sampling method suited for multi-server clusters. Sampling methods mainly focus on manipulating the whole graph structure. In multi-server clusters, we found that sampling the intra-server edges (dependencies) does not bring much speedup but could harm the convergence, as those edges are communicated through the high-speed intra-server links. Therefore, expansion-aware sampling targets to sample only inter-server boundary and can mitigate the neighbor explosion with lower error.

Lastly, we introduce one-hop graph masking to realize a framework that considers multi-server clusters while supporting both full-batch and mini-batch training. By server-wise subgraph extraction and dependency masking with one-hop graph-based data structure, GraNNDis can support the above-proposed methods with high throughput. Additionally, it aids cooperative batching to orthogonally support existing sampling methods.

With the proposed strategies, we evaluated GraNNDis on multi-server clusters. GraNNDis showed significant training throughput increases and comparable/lessened sampling errors compared to prior arts.

---

[1]GraNNDis comes from the Latin word *grandis* meaning 'large.' It is also a combination of Graph, NN, and Distributed.

: Feature | gradient ⬌: External server comm. ⬌: Internal server comm.

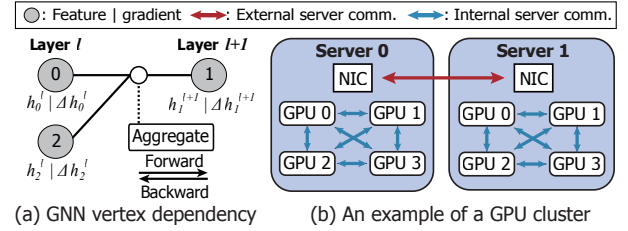(a) GNN vertex dependency   (b) An example of a GPU cluster

Figure 1: (a) An example of vertex dependency on neighbor vertices. (b) A multi-server cluster environment.

In summary, this paper makes the following contributions:

- We identify the advantage of utilizing the characteristics of multi-server clusters to address the challenges in distributed GNN training.
- We propose flexible preloading, which minimizes the inter-server communication bottleneck by utilizing the fast intra-server links.
- We present cooperative batching, which significantly reduces the redundancy issue in mini-batch training by aggregating multiple GPUs in a server.
- We suggest expansion-aware sampling, a sampling strategy that considers the characteristics of multi-server clusters.
- We introduce one-hop graph masking; a computing and communication structure for training in multi-server clusters, to implement the above contributions.
- We evaluated GraNNDis extensively in multi-server clusters to show the speedup and usability compared to prior work.

2 BACKGROUND

2.1 Graph Neural Network (GNN) Training

GNNs [34, 74, 82] show powerful potential for graph representation learning, and as such they are widely used for diverse tasks [12, 39–41]. While transformer architectures [73, 88] have been widely used recently, GNNs are lighter and more specialized in learning graphs' topological characteristics than others. Therefore, GNNs are preferred in real-time workloads [5, 59], data with irregular connectivity (e.g., social networks [15]) and molecular dynamics [2, 18, 20]. The advantages of GNNs can also be exemplified by a trend to utilize GNNs and transformers together [43, 44, 57, 81]. Modern GNN structures most commonly use neighbor dependency information ($\mathcal{N}$) to generate the final representation. Because of this, GNNs recursively require neighbor vertices' hidden vectors of the previous layer to generate the next representation of vertex v. This is called *vertex dependency* in GNN training, as depicted in Figure 1a. Every layer (l) of GNN generates the next hidden vector h_v^l for each target vertex v by aggregating hidden vectors of v's neighbors $\mathcal{N}(v)$, multiplying weights of layer (W^l), and lastly applying activation function (σ). The aggregated vector ($h_{\mathcal{N}(v)}^l$) is often combined with the previous hidden vectors of target vertices h_v^{l-1}. This can be formulated as follows:

$$h_{\mathcal{N}(v)}^l = AGGREGATE(\{h_u^{l-1} | u \in \mathcal{N}(v)\}), h_v^l = \overrightarrow{F}^l(h_v^{l-1}, h_{\mathcal{N}(v)}^l),$$

where $AGGREGATE(\cdot)$ is an aggregation function, usually the summation of all the neighbor hidden vectors. $\overrightarrow{F}^l(\cdot)$ comprises the multiplication of weights (W^l) followed by the activation function (σ).
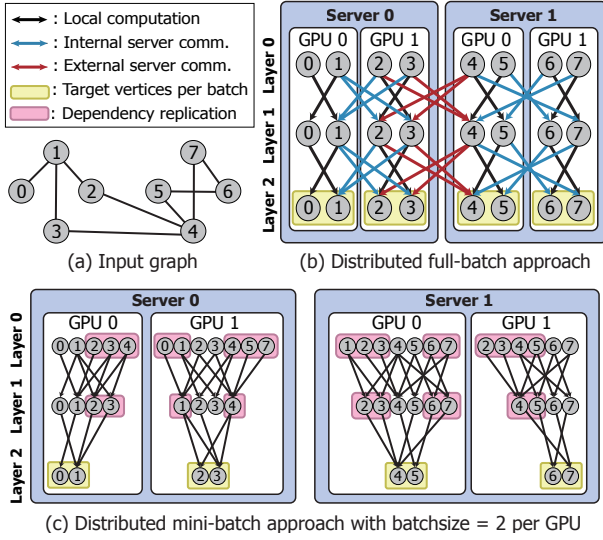
Figure 2: Forward-pass of a 2-layer GNN with two servers, each with two GPUs. (a) An example of an input graph. (b) An example of a full-batch approach. (c) An example of a mini-batch approach with batch size=2 per GPU.

This processing of dependency on neighbor vertices makes GNNs unique compared to other neural network models [35, 67]. Most GNNs mainly suffer from this aggregation process of intermediate representation, thus causing a severe slowdown in training [98].

Traditional GNNs [23, 34] often use three to five layers, but deeper GNNs [40, 41, 49] with residual connections show improved accuracy in various areas. Deep GNNs cause over-smoothing [53, 85], but many works [40, 41, 49] minimize it through normalization/residual connections, achieving higher accuracy [18, 26]. In such deep GNNs, the training becomes much harder because they often use tens or hundreds of layers, requiring much higher memory requirements and much heavier computations. A representative problem is a *neighbor explosion*, which is a result of the increased *Message Flow Graph* (MFG) size by the number of layers. It is becoming more common to use multiple accelerators (e.g., GPUs) to address such challenges.

2.2 Distributed GNN Training

Two representative distributed training methods are used to mitigate the outbursting memory usage of GNN training: full-batch and mini-batch training. Full-batch training is memory-hungry but reaches high final accuracy, while mini-batch training requires less memory and converges faster, but the final accuracy could be lower. Thus, both approaches are widely used.

Full-batch training [28, 76, 77, 80] uses the whole graph representation for every parameter update. As depicted in Figure 2b, all vertex dependencies must be fully computed. Therefore, an iteration (model parameter update) essentially becomes an epoch. For this, all the hidden vectors for every layer have to be loaded on the GPUs' memory, so multiple GPUs are typically required for large graphs [26, 37] and deep GNNs [40, 41] in a distributed manner.

In distributed full-batch training, inter-GPU communications are essential for retrieving intermediate vertex representations from other GPUs, as illustrated by the **blue** and **red** arrows in Figure 2b. The overhead from this communication is significant, so it generally uses a partitioning algorithm [32] to minimize the communication volume when partitioning a graph for GPUs. However, most existing works [28, 47, 94] do not consider multi-server cluster settings and heavily rely on slow external communications. This work provides higher throughput than previous works by considering a characteristic of multi-server clusters that a difference exists between the inter- and intra-communication bandwidth, such as Figure 1b.

Mini-batch training [9, 78, 93, 96, 97] is a technique used when the GPU memory capacity is insufficient for full-batch training. It uses a batch concept to mitigate the memory capacity issue. It packs the partial dependencies into MFGs for the fixed number of target vertices to mitigate the heavy memory overheads, as illustrated with MFGs in Figure 2c. By adjusting batch size, training can be performed with flexible memory requirements, at the cost of redundant memory usage and computation among mini-batches.

Mini-batch training can be easily extended to distributed training [19, 96, 97], as depicted in Figure 2c. Each GPU loads all the vertex dependencies required to compute the feature of target vertices, which means the distributed mini-batch approach does not communicate for fulfilling vertex dependencies between GPUs. After each GPU processes its mini-batch, the weight update is performed collectively by sharing the gradients among the GPUs. In this case, the total batch size is *the mini-batch size × the number of GPUs*. Compared to full-batch training, an epoch requires several iterations to handle all vertices. However, it scales poorly because the GPU-wise mini-batch leads to extreme redundancy in memory usage and computations, as shown in Figure 2c (shaded pink), and demonstrated in Section 3.

2.3 Sampling Methods

Because of the high memory requirements and heavy processing overhead of dependencies in GNNs, many sampling methods [6, 9, 23, 58, 60, 92, 93] are actively being explored. They can be roughly divided into *layer-wise sampling* and *subgraph sampling* approaches. Layer-wise sampling approaches [6, 23, 58] apply sampling algorithm layer by layer, typically constraining the maximum degree of edges called *fanout*. Subgraph sampling approaches [9, 92, 93] sample the subgraph from the original graph and execute forward and backward propagation on the sampled subgraph only. Mini-batch training approaches [19, 78, 96, 97] are commonly merged with sampling methods to minimize their redundancy. These sampling methods manipulate the whole graph structure and could be unsuitable for graphs with a high average degree. GraNNDis introduces a new sampling method advantageous for high average degree graphs considering the characteristics of a multi-server cluster environment.

3 MOTIVATIONAL STUDY

In the previous sections, we discussed existing distributed GNN training methodologies and their limitations. Table 1 directly summarizes the prior arts compared to GraNNDis. GraNNDis is an internal/external server communication-aware and redundancy-aware framework. Additionally, through one-hop graph masking, GraNNDis supports both full-batch and mini-batch training.

Table 1: Comparison of GraNNDis to Prior Art

Method	Multi-Server Cluster Aware	Full-Batch (FB)		Mini-Batch (MB)		Support (Both?)
		Comm. Aware	Int./Ext. Aware	Multi-Server	Batch Limit (Redundancy Aware)	
CAGNET [72]	✗	✗	✗	-	-	FB (✗)
Sancus [56]	✗	✓	✗	-	-	FB (✗)
PipeGCN [77]	✗	✓	✗	-	-	FB (✗)
BNS-GCN [76]	✗	✓	✗	-	-	FB (✗)
PaGraph [45]	✗	-	-	✗	a GPU (✗)	MB (✗)
DSP [4]	✗	-	-	✗	a GPU (✗)	MB (✗)
P³ [19]	✗	-	-	✓	a GPU (✗)	MB (✗)
MariusGNN [75]	✗	-	-	✗	a GPU (✗)	MB (✗)
WholeGraph [86]	✗	-	-	✓	a GPU (✗)	MB (✗)
Quiver [68]	✗	-	-	✓	a GPU (✗)	MB (✗)
SALIENT++ [30]	✗	-	-	✓	a GPU (✗)	MB (✗)
GraNNDis (Proposed)	✓	✓	✓	✓	Multiple GPUs (✓)	FB+MB (✓)

In the present section, we show a motivational study to emphasize the key insights of GraNNDis. Figure 3 shows the result of a motivational study, where the limitations of current distributed full-batch [77] (FB) and mini-batch [30] (MB) GNN training can be observed. We trained shallow (3-layer) from deep (28/56/112-layer) GNNs with residual connections [40] on various datasets. All GNN training is conducted on a four-server cluster, where each server has four RTX A6000 GPUs. For the detailed setup, please refer to Section 8.1.

Insight #1: **Some works considered multi-server environments but with naive expansion without considering the intra-/inter-server bandwidth gap.** Figure 3a shows the training time breakdown of the state-of-the-art full-batch training framework [77] when increasing the number of servers with the Reddit dataset. We break down the training time into four: computation, inter-server communication, intra-server communication, and parameter gradient synchronization. Ideally, using multiple servers should speed up the training, but due to the inter-server communication bottleneck, distributed training is unscalable and even slows down. This is because inter-server communication is much slower than intra-one, and the currently used full-batch GNN training frameworks do not consider this difference.

Insight #2: **Mini-batch-based works suffer from multi-server unsuitable structure (i.e., MFG).** As illustrated in Figure 3a, training deep 112-layer setting faces out-of-memory (OOM). A representative strategy for addressing this issue is mini-batch training. However, mini-batch training suffers from significant redundancy even in the state-of-the-art distributed mini-batch training framework [30], which invalidates the strategy. Figure 3b illustrates such redundancies in various models with different numbers of layers on the Reddit dataset. We applied GraphSAGE [23] sampling with 25 fixed fanouts. Even though the sampling [23] is adopted, massive redundancy still exists due to the neighbor explosion. We started from a batch size of 1K per GPU in a 3-layer case, but with 56 layers, the batch size of only one vertex consumes more memory than the full-batch case, and no batch size fits with 56 layers, even with a minibatch size of 1. There are multiple reasons, but we find the primary reason to be the redundancy caused by the mini-batches being formed in individual GPUs.

Insight #3: **Existing samplings (layer-wise/subgraph) mostly target single GPU environments, which incurs unnecessary information loss.** In Figure 3a, we can figure out that

(a) FB training time breakdown (b) MB redundant memory usage
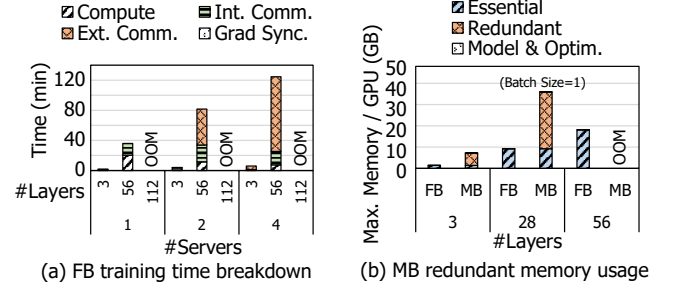
Figure 3: Motivational experiments.

the communication time of inter-server edges (dependencies) consumes a significantly larger portion than the intra-server ones. This implies that sparsifying (sampling) the intra-server edges does not contribute much to a speedup, however, existing samplings do not consider this characteristic from multi-server clusters.

4 NOTATIONS AND PERFORMANCE MODEL

For analysis, we set some notations to explain the latency model of distributed GNN training. N_s and N_g indicate the total number of servers and the number of GPUs per server, respectively. V means the total number of vertices in a given graph and E is the number of edges. C is the computational throughput measured in vertices per second. B_{intra} and B_{inter} are intra-/inter-server bandwidths split per GPU, which are also measured in vertices per second. L is the total number of GNN layers, which means that the model extracts the information from L-hop neighbors. In the analysis, we assume that hidden dimension sizes and vertex feature sizes are equal in every layer for convenience.

We formulate the training time of the previous full-graph training method (T_{prev}) as follows, concentrating on the communication of the vertex dependencies that cannot be overlapped.

$$T_{prev} = \frac{V}{N_s N_g C} + \frac{E}{N_s N_g}\left[\frac{((N_g-1)/N_g N_s)}{B_{intra}} + \frac{(1-1/N_s)}{B_{inter}}\right]. \quad (1)$$

The first term of Equation (1) presents the total compute time of the previous full-graph training. The second term calculates the total communication time, which is the summation of intra- and inter-server communication time. One observation from the communication term is that inter-server communication has larger numerators (more inter-server GPUs) and smaller denominators (lower inter-server bandwidth). This makes inter-server communication become the bottleneck under the latency model. The parameter gradient synchronization by all-reduce communication is omitted because it is almost entirely overlapped by computation through gradient bucketing [55].

5 GRANNDIS FRAMEWORK

Using the characteristics of multi-server clusters, we introduce three schemes of GraNNDis to address the issues from the motivation study. We propose methods to gain speedup under memory budget for full-batch training (Section 5.1) and to generally support diverse graph sampling methods [23, 93] on large batches for mini-batch training (Section 5.2). In addition, we suggest another novel sampling method for multi-server clusters, which gives great efficiency in speed and achieved accuracy (Section 5.3).
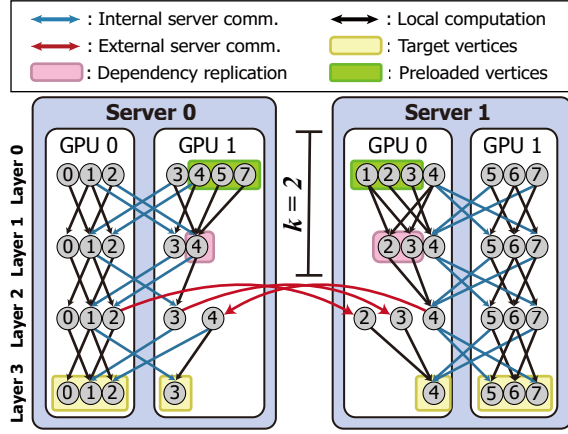
Figure 4: Flexible preloading mitigates the inter-server communication bottleneck using fast intra-server links. The green area represents the preloaded initial graph features.

5.1 Flexible Preloading

> • **Key Idea**: Minimize inter-server communication within the memory budget in full-batch distributed training.

In Section 3, we showed that previous distributed GNN training methods suffer from low scalability from not considering the inter-/intra-server bandwidth difference of multi-server clusters. We propose *flexible preloading* for full-batch training, which considers this difference to reduce inter-server communications. As discussed earlier, the inter-server connections (e.g., Ethernet, Infiniband) are typically an order of magnitude slower than the intra ones (e.g., PCIe, NVLink) in multi-server clusters. Therefore, it is important to avoid using inter-server connections as much as possible.

To minimize inter-server communication, the vertex dependencies should be handled in another way. Flexible preloading uses a preloading strategy to fulfill such dependencies. We can choose to include the remote vertices in the extracted graph (replication) instead of having to communicate, similar to mini-batching methods [96, 97]. However, the power-law distributions of real-world graphs [37] would easily cause the extracted graph to be very large, which diminishes the point of distributed training from a memory capacity perspective.

Flexible preloading attempts to avoid such problems but still minimizes inter-server communications, as depicted in Figure 4. Starting from layer 0, the same one-hop graph structure is used for a set of consecutive k layers applying one-hop graph masking, which will be discussed in Section 6. On the next layer (i.e., layer $L − k + 1$), the one-hop graph is reshaped, which is then reused in the succeeding layer. This reduces inter-server communication by replicating vertices at a minimal amount of reshaping cost. We choose the largest k to make the GNN fit on the available memory.

Figure 4 illustrates an example where $k = 2$ is used for a 3-layer GNN. The vertices are split into two servers such that vertices 4-7 belong to server 1 (the inner vertices of server 1). Note that we use imbalanced partition among GPUs for a clear view, but balanced partitioning [32] is used in practice, because the neighbors expand in all directions in real-world graphs. one-hop graph masking is used but omitted from the illustration. In layer 0, one-hop neighbors

are preloaded (shaded green), reducing the inter-server communication. For layers from one to $k − 1$, one-hop graph masking is needed to adapt to consider the exact dependency. After layer $L − k + 1 = 2$, a new (smaller) one-hop graph is used, which allows inter-server communications (vertex 2, 3, and 4) in the subsequent layers. However, most communications are intra-server, not severely slowing down the system.

Flexible preloading incurs some redundant computation across servers as a cost for reduced inter-server communication. However, it is advantageous because the devices' computational throughput far exceeds that of the inter-server communication in recent multi-server clusters. We approximately formulate such a trade-off with a performance model. From Equation (1), we can derive the training time and the expected speedup using flexible preloading. Here, we assume that the dependency graph size grows at the rate of D^α every layer, where D is the average degree of vertices, and $0 < \alpha < 1$ is a graph-dependent per-layer expansion factor to represent overlapping neighbors. Also, we set $k = L$ for brevity. The inter-server bandwidth (B_{inter}) is replaced by the computation of preloaded vertices. In addition, the preloaded vertices increase the amount of intra-server communications. With these considerations, the training time changes as follows:

$$T_{preload} = \frac{VD^\alpha L}{N_s N_g C}, + \frac{ED^\alpha L}{N_s N_g} \left[\frac{(N_g - 1)/N_g}{B_{inta}} \right], \quad (2)$$

$$1 < D^\alpha L \le N_s N_g. \quad (3)$$

The condition for $T_{preload}$ to be shorter than T_{prev} from Equation (1) is $T_{preload} − T_{prev} < 0$. Making some rough approximations as $1/B_{internal} \approx 0$ and $1/N \approx 0$, the condition becomes

$$\frac{VD^\alpha L}{N_s N_g C} - \frac{V}{N_s N_g C} - \frac{E}{N_s N_g} \frac{1}{B_{inter}} < 0. \quad (4)$$

Because $E = V \cdot D$ by definition,

$$\frac{C}{B_{inter}} > \frac{(D^\alpha L - 1)}{D}. \quad (5)$$

Equation (5) suggests how much gap is needed between computational throughput and communication bandwidth for flexible preloading to gain speedup. Using $D^\alpha = 2$ and $D = 10$ for a three-layer GNN ($L = 3$), flexible preloading has an advantage when the processing rate (vertex / second) of computation is at least half that of the inter-server communication, which is the usual case in cluster environments.

5.2 Cooperative Batching

> • **Key Idea**: Reduce redundancy and support sampling techniques in mini-batch distributed training.

In mini-batch training framework, the vertex dependencies are fetched to individual GPUs. This strategy results in a lot of redundant memory usage and greatly limits the size of a mini-batch, as depicted in Figure 1c. We propose cooperative batching to mitigate such limitation of the previous strategy with the aid of one-hop graph masking, which will be discussed in Section 6. With cooperative batching, to suppress redundancy and enable a much larger batch size, we cooperate the GPUs within a server as if they were
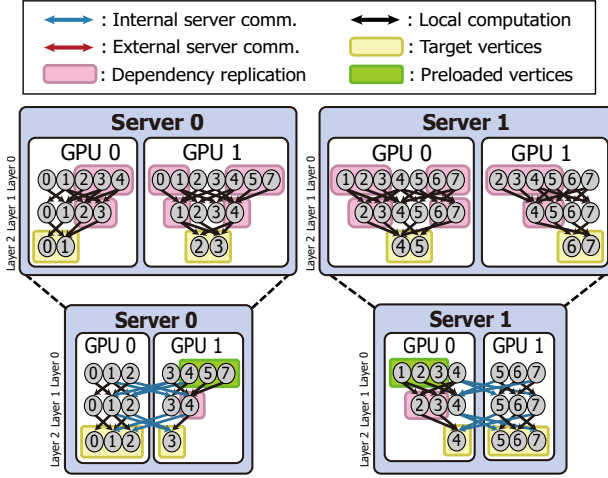
Figure 5: In the conventional mini-batch approach, the memory size of a single GPU bounds the mini-batch size. Cooperative batching enables larger batches while reducing redundancy using fast intra-server links.

Figure 6: Expansion-aware sampling applies a sampling on the server boundary to mitigate the neighbor explosion.

a single large GPU, as illustrated in Figure 5. We choose the target vertices to form a *co-batch* and form an MFG of them. While existing mini-batch training frameworks fetch MFGs to each GPU, cooperative batching fetches co-batches to each server. Then, each co-batch is distributed to the internal GPUs of each server.

The application of cooperative batching has a clear advantage of substantially less redundant memory and, thus, a larger batch size. It also significantly reduces the redundant computation of mini-batch training. By enlarging a batch size limit, we could address the outbursting memory usage of recent GNNs due to an increase in the number of layers and graph size growth. The disadvantages are increased intra-server communication and the possibility of an imbalanced workload between workers. However, neither is a severe overhead because the intra-server bandwidth is very high, and the redundancy is significantly reduced.

One remaining issue is supporting sampling techniques. As sampling approaches [23, 93] are almost de facto standard to mini-batch training, cooperative batching also supports existing sampling methods by one-hop graph masking. For layer-wise samplings [6, 23, 58], the maskings are generated per layer to conduct layer-wise aggregation of them. While existing subgraph sampling methods [9, 93] construct subgraphs in a GPU-wise manner, they are identical to the unmasked version of one-hop graph masking when extending them to a server-wise manner, which will be described in Section 6.

5.3 Expansion-aware Sampling

- **Key Idea**: Apply sampling only in the expansion region that affects the system speed.

As discussed in Section 2, sampling methods [9, 23, 58, 60, 93] are popular techniques in both full-batch and mini-batch training, which aim to prevent the neighbor explosion [9, 37, 76]. Meanwhile, the implication behind the huge speedup of flexible preloading and
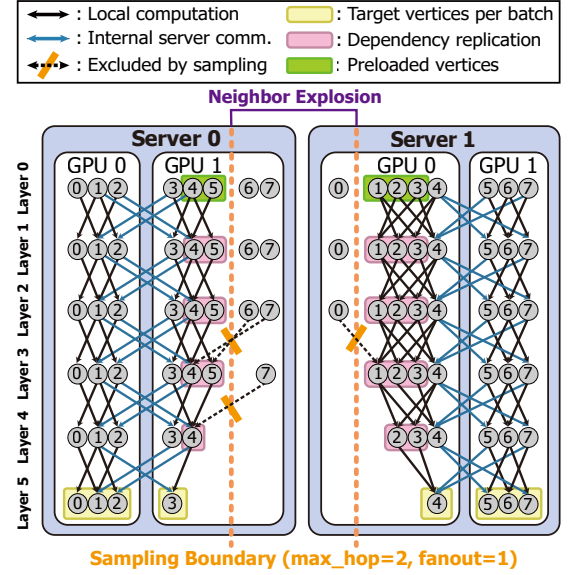
cooperative batching is that computation and intra-server communication are at least several times faster than inter-server communication. Considering these, we observe that the sparsification of dependencies inside a GPU, or among GPUs in the same server does not contribute much to the speedup while they lose the same amount of information in a graph. In such regard, prior sampling methods could be unsuitable for graphs with high average degrees (e.g., Reddit, average degree 492) because they sample the whole graph structure (e.g., with fanout 25).

Therefore, we propose expansion-aware sampling, which aligns with the lessons from the performance implications in multi-server clusters. Instead of sampling the whole graph structure, expansion-aware sampling only targets to sample the server boundary vertices in a multi-server cluster. Only from the server boundary vertices, we define maximum hops (h) and maximum fanouts (f) towards the external vertices in remote servers for sampling, and maintain all other dependencies (i.e., inner vertices) untouched. As we will show, it adds a great amount of speedup compared to other methods, at a comparable or better accuracy.

5.3.1 Sampling Details. Figure 6 illustrates the detailed sampling methodology of expansion-aware sampling. For simplicity and visibility, we assume that vertices 0-3 and vertices 4-7 are dedicated to servers 0 and 1, respectively. When the vertices for each server that are not in inner vertices are preloaded for full-/mini-batch training, expansion-aware sampling applies constraints to decide whether a certain vertex is included.

We provide an example of $h = 2$ and $f = 1$. For expansion-aware sampling, we select external vertices that are at maximum a two-hop distance ($h = 2$) from the server's inner vertices. Moreover, expansion-aware sampling restricts the hop traversing to find only one external neighbor ($f = 1$). When applying the fanout, to consider the importance of each vertex, expansion-aware sampling selects the vertices that have many connections (degrees) to the unselected vertices. Therefore, the vertices from 4 to 7 are to be

preloaded by server 0 without any sampling, but expansion-aware sampling only decides to preload the vertices 4 and 5. When the traversal is finished, a subgraph is generated from the traversed vertices, and the internal dependencies of the inner vertices remain unsampled as a whole. We empirically used the settings of $h = 1$ and $f = 15$, which aligns with prior works [17, 61] that emphasize the importance of 1-hop dependency. We observe that the number of inner vertices (at least 10K) is sufficiently large, and the min-cut algorithm (i.e., METIS [32]) for dedicating inner vertices helps the sampling to contain the whole graph's structural information. We provide the sensitivity of the tunable parameters in Section 8.5. The cost of expansion-aware sampling is formulated as below:

$$T_{sampling} = \frac{Vm^\alpha k}{N_s N_g C} + \frac{Em^\alpha k}{N_s N_g} \left\lceil \frac{(N_s N_g - 1)}{B_{intra}} \right\rceil, \tag{6}$$

$$1 < m^\alpha k \ll D^\alpha L \le N_s N_g. \tag{7}$$

While this preserves the number of edges in the perspective of the whole graph, it has the effect of reducing the growth to $m^{\alpha k}$, much smaller than $D^{\alpha L}$ since $m \ll D$ and $k \ll L$.

5.3.2 Approximate Error Bound Analysis. We approximately compare expansion-aware sampling with the layer-wise sampling [23] and the subgraph sampling [9] as follows. By comparing the approximate error bounds, we found that expansion-aware sampling provides a smaller error than the previous subgraph sampling method [9] and is advantageous for high average-degree graphs than the layer-wise sampling [23].

Error bound of samplings. We apply Theorem 2 of [17] for the average pooling and replace historical error terms (ϵ) with sampling error terms (ϵ_{samp}) as the historical and sampling [23, 93] errors both affect the node embeddings in each layer. emb_v^L is a full-batch embedding of v after passing L layers. The closeness of the approximated embedding to the full-batch embedding can be represented by $||\tilde{emb}_v^L - emb_v^L|| \geq 0$. The error of the sampled embedding from the approximated embedding for each layer l is $0 \le ||\tilde{emb}_{v,sampled}^{(l)} - \tilde{emb}_v^{(l)}|| \le \epsilon_{samp}^{(l)}$. Let $AGGREGATE(\cdot)$ and $\overrightarrow{F}^l(\cdot)$ be Lipschitz continuous with constants c_1 and c_2, and set $c_1 c_2$ as C. According to [17], the final error of a vertex v is approximately bounded by

$$||\tilde{emb}_v^L - emb_v^L|| \lesssim \Sigma_{l=0}^{L-1} \epsilon_{samp}^{(l)} C^{L-l}. \tag{8}$$

The error from a layer $(l-1)$ propagates to the next layer (l) with the constant C. Therefore, C^{L-l} intuitively shows that the errors from earlier layers propagate and accumulate to deeper layers.

Layer-wise sampling [23]. We can set the error of the layer-wise fanout sampling as ϵ_{fanout} and replace ϵ_{samp} of Equation (8). $|V|$ indicates the total number of vertices. When we set the average error among vertices from a fanout sampling as $\bar{\epsilon}_{fanout}$, the total maximum error bound of all vertices using a fanout sampling method can be calculated as follows using Equation (8):

$$ERR_{fanout}^{max} = \Sigma_{l=0}^{L-1} \bar{\epsilon}_{fanout}^{(l)} |V| C^{L-l}. \tag{9}$$

Expansion-aware sampling (Ours). By changing Equation (9) for expansion-aware sampling, we can additionally figure out the importance of not sparsifying the dependencies among GPUs in the same server. GraNNDis generates a huge batch at a server level. In

this case, a relatively small portion of inner vertices requires outer-boundary vertices at each hop. To consider such an effect of a large batch, we can introduce coefficients α and β ($\alpha + \beta = 1$). α is the ratio of the total number of inner vertices not requiring outer-boundary vertices to the number of preloaded vertices ($\alpha > 0$): therefore, α becomes large as batch size increases. We set the error from an inner vertex as ϵ_{in}, which is close to 0 because inner vertices do not use sampled information for each hop. Meanwhile, β is the ratio of the number of boundary vertices to the number of replicated vertices ($\beta > 0$). We set the error from a boundary vertex as ϵ_{out}, which is a small value with the aid of a min-cut algorithm such as METIS [32]. When setting the coefficients and errors to derive the maximum error bound of expansion-aware sampling in Equation (9),

$$ERR_{EAS}^{max} = \Sigma_{l=0}^{L-1} (\alpha^{(l)} \bar{\epsilon}_{in}^{(l)} + \beta^{(l)} \bar{\epsilon}_{out}^{(l)})|V| C^{L-l} \simeq \Sigma_{l=0}^{L-1} \beta^{(l)} \bar{\epsilon}_{out}^{(l)} |V| C^{L-l}. \tag{10}$$

β is roughly inversely proportional to the batch size (or partition size), and therefore the error is small for larger batch sizes. However, most existing sampling, such as fanout sampling (Equation (9)), has no such effects because it naively restricts the fanout of *all vertices*. We compare the actual error of expansion-aware sampling (ERR_{EAS}^{actual}) with the fanout sampling (ERR_{fanout}^{actual}) and analyze the empirical values of β in Section 8.8.

Subgraph sampling [9]. We can apply Equation (10) to subgraph sampling methods [9, 92, 93] that use a smaller GPU-wise subgraph mini-batch, because expansion-aware sampling can be seen as a generalization of subgraph sampling. In those cases, the mini-batch size becomes extremely small (a hundred to a few thousand), so β is close to 1 ($\beta \simeq 1$). Also, it drops more outer boundary information, so $\bar{\epsilon}_{subg}$ would be larger than $\bar{\epsilon}_{out}$. Therefore, the above makes the maximum error always larger than that of expansion-aware sampling as follows:

$$ERR_{subg}^{max} \simeq \Sigma_{l=0}^{L-1} (1 \cdot \bar{\epsilon}_{subg}^{(l)})|V| C^{L-l} > ERR_{EAS}^{max}. \tag{11}$$

We also compare the actual errors of the subgraph sampling method (ERR_{subg}^{actual}) to other methods in Section 8.8.

6 ONE-HOP GRAPH MASKING

> • **Key Idea**: Support both full-/mini-batch training to efficiently use intra-server links of multi-server clusters.

The methods of GraNNDis utilize the fast intra-server bandwidth of recent multi-server clusters while avoiding the slower inter-server communications on both full-batch and mini-batch training. As conceptual methods, we propose flexible preloading and cooperative batching. However, realizing such a unified framework with existing work is difficult, because of the underlying difference in the software structure. Therefore, we introduce one-hop graph masking, an efficient data structure, and software architecture to realize it.

Figure 7a depicts the commonly used structure of a state-of-the-art full-batch training [76, 77]. They are based on a one-hop graph structure (src→dst) to minimize random accesses and avoid frequent graph reshaping. Between each layer, the vertex features are fetched from remote GPUs, either by inter- (red) or intra- (blue) server communications. To avoid heavy random access overhead, a separate contiguous memory region is assigned for every peer
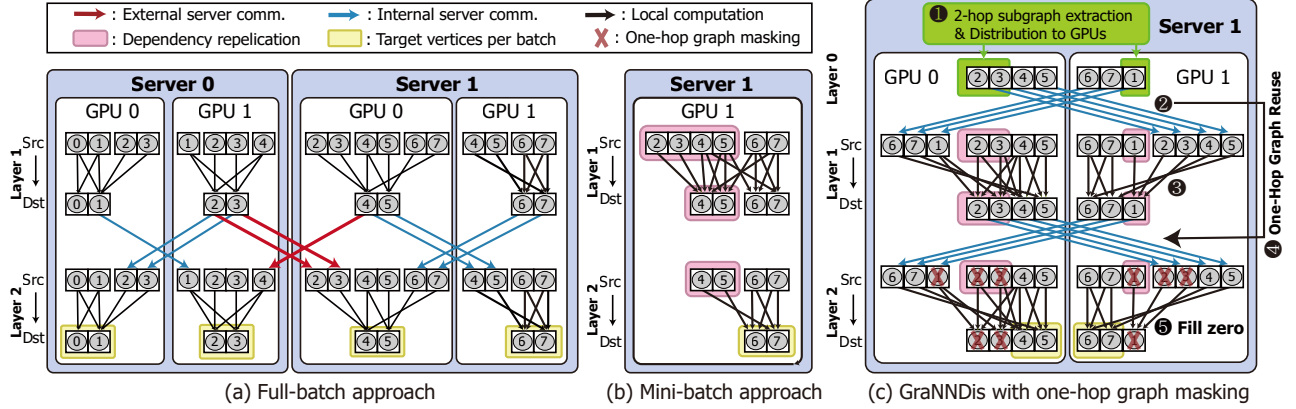
Figure 7: Baseline approaches and overall structure of GraNNDis with one-hop graph masking on a multi-server cluster.

GPU. Then, the required vertex features are transferred among GPUs in an all-to-all [70] pattern, where the vertices are locally sorted. This way, the accesses are piecewise sequential in both the communication and computation. Furthermore, the same one-hop graph structure is used in every layer, avoiding repetitive reshaping operations.

On the other hand, mini-batch training frameworks [30, 31, 96, 97] take a completely different approach as shown in Figure 7b. Because all dependent neighbors are included in the MFG, there is no structure for communication. Furthermore, each layer has a differently shaped graph, which prohibits naive adoption of one-hop graph structures from Figure 7a. While it is doable in principle, switching to different subsets of vertices with distinct ordering would cause severe latency overhead.

To address aforementioned issues, GraNNDis uses a novel strategy of *one-hop graph masking*. The key idea is to reuse the same one-hop graph structure, but apply different masking every layer to support varying vertex subsets and dependency handling. This approach takes the benefit from both sides of full-batch and mini-batch training. The partially sequential accesses and graph reuse property of the full-batch training are preserved, while the effective structure can freely vary with high flexibility.

Figure 7c shows the procedure and structure of GraNNDis with one-hop graph masking based on the example graph. ❶ As the first step, it extracts selected neighbors from multi-hop dependency starting from the inner vertices of a server. Then, the extracted vertices are distributed to the GPUs in a server. For example, if we choose to extract entire 2-hop neighbors from the inner vertices 4-7, the vertices 1-7 are extracted and partitioned to GPUs (vertices 2-5 to GPU 0 and the rest to GPU 1) ❷ For each layer, similar to the distributed full-batch training (Figure 7a), each GPU receives the required vertex features. In this example, because we chose 2-hop neighbors for the 2-layer GNN, only the intra-server communication remains, and all of the slow inter-server dependencies are already replicated in the extracted graph. ❸ Then, the next vertex features are computed from the received features. ❹ In the next layer, the same one-hop graph and communication pattern are reused. After then, with one-hop graph masking, we generate a mask for the unneeded vertices, which fills zeros to the features. ❺ As a result, we can generate the final outputs (logits) of the inner vertices considering the exact dependency.

7 IMPLEMENTATION DETAILS

We implement GraNNDis based on vanilla [77] which significantly improves the throughput of [28] and achieves state-of-the-art performance. It uses GLOO [14], which does not support GPU RDMA features; therefore, we reimplemented it with NCCL [51]. Further, while the original method [77] uses custom parameter gradient synchronization, we modified it to use the PyTorch DistributedDataParallel (DDP) package to provide overlapping gradient synchronization. These optimizations show significant speedup, so we set this optimized version [77] as our full-batch training baseline.

For parallelization, GraNNDis uses subgraphs generated by flexible preloading and cooperative batching according to the batching scheme (full-/mini-batch). The subgraphs are distributed to each server, where one-hop graph masking is utilized for intra-server parallelization among GPUs. One-hop graph masking further creates subgraphs for preloading and partitions the server-dedicated subgraphs. For this purpose, we make a 'head group' comprising the first process of each server, and this head group handles those procedures. Additionally, GraNNDis does not employ pipelining because pipelining for GNN training often indicates asynchronous staleness with lower accuracy and is seldom utilized in practice.

8 EVALUATION

8.1 Experimental Environment

Cluster environment and model configuration. Table 2 lists our cluster environments. We use clusters equipped with four RTX A6000 GPUs, an AMD EPYC 7302 CPU, and a 512GB system memory. Our cluster has Infiniband QDR (32 Gbps) for external connections between servers, and NVLink is used for internal connections. We trained GraphSAGE [23] for shallow GNNs as a default. For deep GNNs, we used ResSAGE+ [41] with GraphSAGE as a graph convolution. We also trained GCN [34] for some evaluations. Note that for full-batch training, GraphSAGE and ResSAGE+ do not use the fanout sampling of GraphSAGE but only adopt the model structure. We used 64 as the default hidden dimension size in all experiments.

Datasets. We chose six datasets; ogbn-arxiv (Arxiv, AX), ogbn-products (Products, PD), Reddit [84] (RD), igb-small (IGB (S)), igb-medium (IGB (M)) [33], and ogbn-papers100M (Papers) [26]. The characteristics of datasets are summarized in Table 3. Arxiv is a citation network where each edge indicates a source paper citing
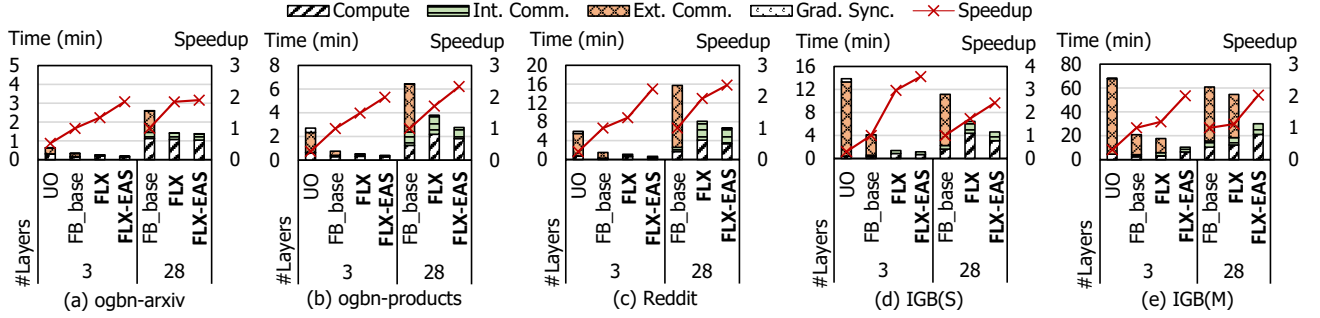
Figure 8: Full-batch training time breakdown and speedup of the unoptimized/optimized baselines (UO, FB_base), flexible preloading (FLX), and expansion-aware sampling (FLX-EAS) in shallow to deep GNNs.

Table 2: Experimental Environment

HW	Server	GPU	4× NVIDIA RTX A6000
		CPU	1× EPYC 7302, 16C 32T
		Memory	512GB DDR4 ECC
		Int. connect	NVLink Bridge [52]
	Cluster	#Servers	4
		Ext. connect	Infiniband QDR (32Gbps)
SW	Common	Python	3.10
		PyTorch	1.13
		CUDA	11.6.2
		NCCL	2.10.3
	GNN	DGL	0.9.1 [78]
		Model	GraphSAGE [23], ResSAGE+ [41], GCN [34]
		Hidden dim.	64
		Task	Node classification

Table 3: Graph Datasets

Size	Name	Dataset Info.			Hyper-parameter		
		#Nodes	#Edges	Feat. size	lr	Dropout	#Epochs
Small~ medium	ogbn-arxiv [26]	0.17M	1.2M	128	0.01	0.5	1000
	ogbn-products [26]	2.45M	61.9M	100	0.003	0.3	1000
	Reddit [84]	0.23M	114.6M	602	0.01	0.5	1000
Large~ Hyperscale	IGB (S) [33]	1M	12.1M	1024	0.01	0.5	1000
	IGB (M) [33]	10M	120.1M	1024	0.01	0.5	1000
	ogbn-papers100M [26]	111M	1.6B	128	0.01	0.5	1000

a destination paper. Products represents a co-purchasing network, and each node represents a product in Amazon. Reddit consists of nodes representing posts in the online forum of Reddit. IGB and Papers are also citation networks whose sizes are orders of magnitude larger than other datasets. We mainly used the hyperparameters chosen in [33, 77]. As Papers is a representative hyper-scale dataset, we dedicated a separate section in Section 8.3.

Baselines. We mainly chose two representative state-of-the-art baselines in our experiments, PipeGCN [77] and SALIENT++ [30]. PipeGCN [77] is a state-of-the-art full-batch training framework. We used the vanilla version of [77] (i.e., without the pipeline that involves staleness), which is essentially a refactored version of existing full-batch training. We optimized the vanilla [77] with NCCL, showing a much faster training time than the original version. We set this as a full-batch training baseline and denoted it as FB_base. For the pipelined [77], we tested it together with other full-batch-based staleness [56] and sampling [76] methods in Section 8.7. SALIENT++ [30] is a state-of-the-art distributed mini-batch training framework, and it mandates neighborhood sampling in each training iteration. It significantly improves the training throughput

of DistDGL [96, 97] with sampling/computation overlapping and aggressive caching. We used SALIENT++ as a mini-batch training baseline and denoted it as MB_base. For further validation, we also reported the throughput of DistDGL [96]. However, DistDGL is significantly slower than the others, so we reported it only in Figure 9. We trained all datasets for 1000 epochs with full-batch training baseline [77] and GraNNDis. With mini-batch training baselines, we set the total epoch of 30 and the batch size per GPU as 1024 and weak-scaled it, following the common practices from [30, 96, 97]. We tried to enlarge a batch size of mini-batch training baselines for a fair comparison with GraNNDis. However, it only worsens the sampler overhead of baselines and brings a slowdown to them. Therefore, we settled to follow the optimized practices of their original papers [30, 96, 97]. Note that we applied layer-wise sampling [23] for mini-batch baselines following the common practice.

8.2 Training Speedup and Breakdown

Full-batch training. Figure 8 shows the speedup of GraNNDis in full-batch training of GNNs compared to the unoptimized (UO) and optimized full-batch training baseline [77] with our optimizations (FB_base). We chose 3 and 28 layers, which are widely used as the number of layers for shallow [23, 34] and deep GNNs [40, 41]. For expansion-aware sampling (EAS), we used 1-hop and 15-fanout as default settings if not otherwise stated. We verified the speedup from the optimization discussed in Section 7. Our optimization of using NCCL (UO→FB_base) provides 1.92-4.00× speedup, as illustrated in Figure 8. The unoptimized baseline fails to run deep GNNs due to its misuse of the system resources (i.e., sockets), so we reported its shallow layer case.

In all GNNs from shallow to deep layers, flexible preloading (FLX) provides a speedup over the optimized baseline FB_base, as shown in Figure 8. This speedup comes from minimizing inter-server communication by preloading the vertex dependencies and sharing them within a server. In FB_base, inter-server communication consumes 44.07-85.81% of the total training time. Flexible preloading (FLX) reduces this huge portion and provides 1.11-2.96× speedup. In IGB (M), there is less chance to preload all vertex dependencies due to the memory limit, so flexible preloading provides a relatively smaller, but still significant speedup of 1.19× and 1.11× in 3- and 28-layer GNNs, respectively.

Expansion-aware sampling (FLX-EAS) further accelerates the training by reducing the redundant computation and internal communication from the neighbor explosion. For example, in the Reddit
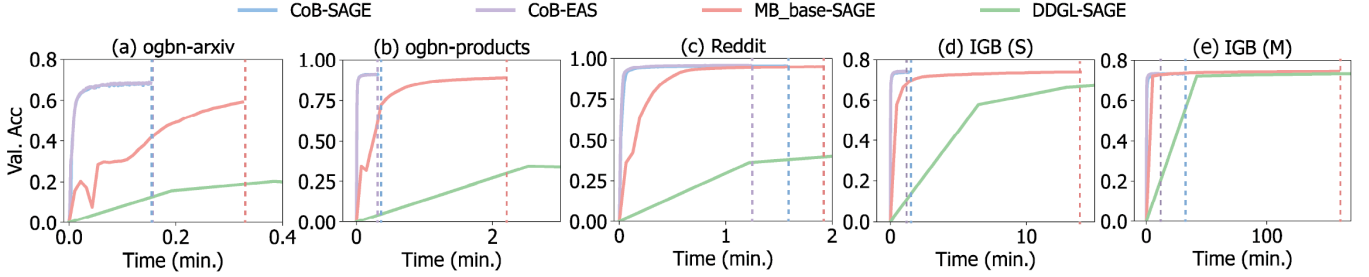
Figure 9: Mini-batch training time-to-accuracy curves of DistDGL (DDGL), SALIENT++ (MB_base), and cooperative batching (CoB) on various datasets. The dotted vertical lines depict the end of training.

dataset with a 28-layer case, sole flexible preloading already provides 1.94× speedup, but expansion-aware sampling addresses the neighbor explosion and shows 2.37× speedup. For other datasets, when expansion-aware sampling is adopted, it provides a significant 1.85-3.56× speedup in shallow to deep GNNs. It is worth noting that applying expansion-aware sampling yields comparable or better accuracy. We further discuss it in Table 4 and Section 8.8.

Mini-batch training. Figure 9 shows time-to-accuracy plots of the proposed cooperative batching (CoB) and SALIENT++ [30] (MB_base), a famous state-of-the-art distributed mini-batch training method. We trained the default shallow model, 3-layer 64-hidden size GraphSAGE. We additionally examined DistDGL [96] (DDGL), another popular distributed mini-batch training framework. As mini-batch training methods generally mandate sampling methods, we adopted the fanout sampling from 25-fanout GraphSAGE [23] to MB_base and DDGL. For a fair comparison, we also applied 25-fanout GraphSAGE to the proposed CoB (CoB-SAGE). Additionally, we reported the time-to-accuracy results when expansion-aware sampling was adopted by CoB (CoB-EAS).

CoB-SAGE provides superior speedup over the baseline MB_base of 1.62-12.69×. DDGL suffers from severe sampler overhead and is much slower than MB_base and cooperative batching. For better visibility, we omitted the full training curve of DDGL. MB_base tends to show much significant redundancy in larger datasets with higher average degrees, so CoB-SAGE provides 12.69× speedup in the IGB (M) dataset. When comparing CoB-EAS with CoB-SAGE, CoB-EAS generally showed faster training time, showing up to 1.27× speedup in the Reddit dataset. We omit the results from deep layers because MB_base and DDGL yield out-of-memory errors either from the GPU or from the sampler in most datasets. As we will show in the next, GraNNDis allows the handling of deep layers.

8.3 Hyper-Scale Dataset and Deeper Models

To observe the scalability of GraNNDis on a hyper-scale dataset, we trained ogbn-papers100M (Papers). The results are shown in Figure 10a. We used 3-layer 64-hidden size GraphSAGE and normalized all results to the training time of SALIENT++ running on a single server. FB_base fails to train from OOM issues, even with 16 GPUs from 4 servers. SALIENT++ (MB_base) is able to train the model with a smaller batchsize and aggressive sampling, but it does not scale well due to large redundancy. On the other hand, GraNNDis at full-batch mode using expansion-aware sampling (FLX-EAS) is able to train the model with good scalability in the number of servers.
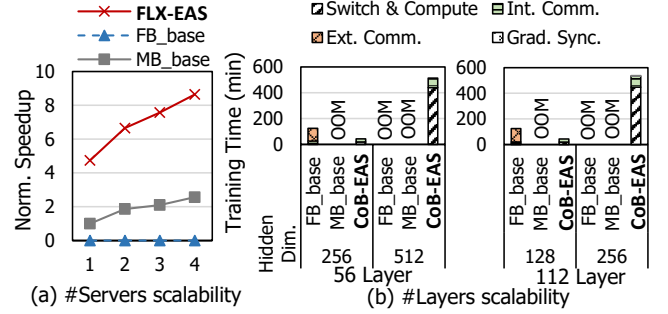

Figure 10: Scalability on the number of servers and layers.

FLX-EAS provides a huge speedup ranging from 3.37× to 4.74× compared to the baseline with the same number of servers. Further, it achieves 60.84% accuracy, while MB_base got 60.58%. This supports that expansion-aware sampling provides stable convergence.

In Figure 10b, we demonstrate the benefit of GraNNDis by providing cooperative batching and expansion-aware sampling on deeper GNNs of 56- and 112-layers with Reddit dataset, and the hidden size ranging from 128 to 512. In our cluster environment, both the FB_base and the MB_base baselines often fail from out-of-memory errors. This is a natural phenomenon in FB_base because it requires enough GPU memory to store all vertex features from all layers. In such cases, MB_base is expected to be an alternative, which allows training with smaller mini-batches. Unfortunately, MB_base failed to train them even at the mini-batch size of 1, despite our faithful effort to save memory space such as checkpointing [55]. On the other hand, GraNNDis generates a batch at a server level with aggregated GPUs, which can mitigate heavy memory overhead of deeper and wider GNNs with less redundancy. As a result, GraNNDis in minibatch mode with expansion-aware sampling (CoB-EAS) is the only choice to train the Reddit dataset in a 112-layer and 256-hidden dimension size with the cluster under test. Moreover, when the model fits the memory with FB_base, GraNNDis provides speedups of 2.92× and 2.91× in 56- and 112-layer GNNs, respectively.

8.4 Mini-Batch Sampling Methods on GraNNDis

As GraNNDis supports most existing sampling methods with cooperative batching, we compare their training time and accuracy implemented on cooperative batching. Figure 11a shows the training time breakdown and accuracy of cooperative batching when applying subgraph sampling (GraphSAINT [93], SAINT), layer-wise sampling (GraphSAGE [23], SAGE), and expansion-aware sampling (EAS). As SAINT removes any neighbors outside the target vertices,
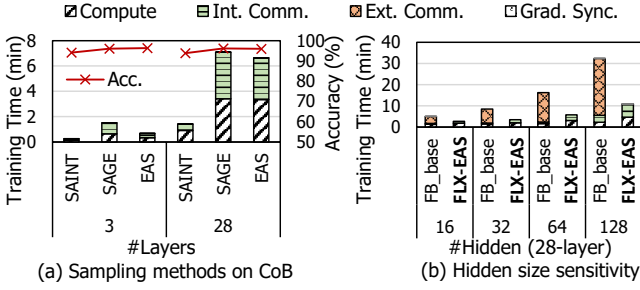
Figure 11: (a) Comparison of sampling methods implemented on cooperative batching. (b) Hidden size sensitivity of GraNNDis (GRD) on Reddit dataset.
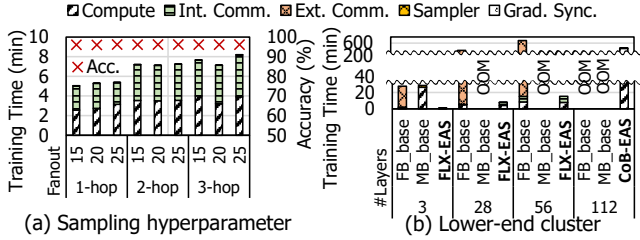

Figure 12: (a) Sensitivity of expansion-aware sampling on hyper-parameter with Reddit. (b) Training time breakdown on a lower-end cluster with Reddit.

it provides a shorter training time but shows some accuracy degradation. On the other hand, SAGE and EAS both consider further hops and fanouts of target vertices and show longer training time while achieving higher accuracy. With cooperative batching, applying GraphSAGE (SAGE) shows significantly less training time than MB_base (in Figure 9) from less redundancy.

8.5 Sensitivity Studies

Hidden dimensions. Figure 11b shows the hidden dimension size sensitivity of GraNNDis on the Reddit dataset with 28-layer GNN. Overall, GraNNDis (FLX-EAS) consistently provides 1.85-2.97× speedup over FB_base. As the hidden dimension size grows, the portion of external communication increases, which leads to more speedup of GraNNDis.

Sampling hyperparameters. Figure 12a illustrates how the hyperparameters of further hop and larger fanout change training time and accuracy of expansion-aware sampling. We tested the 1- to 3-hop settings with a 15- to 25-fanout setting, which are widely used fanout values in sampling methods. All experiments are conducted in the full-batch mode with expansion-aware sampling (FLX-EAS). As we increase the maximum hops, the training time increases up to 2-hop with a 25-fanout setting. On the other hand, all settings show similar accuracy, which means that expansion-aware sampling can reach full accuracy even with 1-hop with a 15-fanout (default).

Lower-end cluster. The ability of GraNNDis being able to free transit between full-batch and mini-batch shines better with low-end hardwares, especially with limited memory. We equipped a lower-end cluster with two servers, each with four RTX 2080Ti (11GB device memory), connected with 1GbE external bandwidth. As illustrated in Figure 12b, in various layer settings with a hidden size 64, GraNNDis provides 29.07-44.64× speedup on Reddit
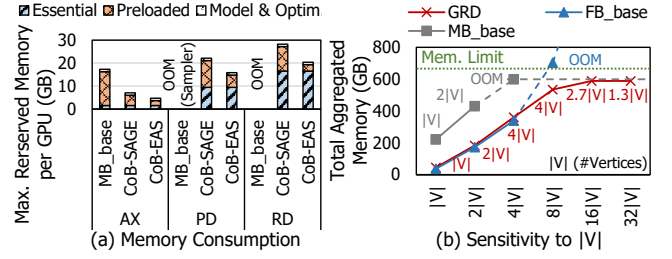

Figure 13: (a) Memory consumption analysis of MB_base and GraNNDis. (b) Memory consumption of the baselines (FB_base, MB_base) and GraNNDis (GRD) when the graph size becomes large.

compared to FB_base and MB_base. MB_base suffers from GPU OOM due to severe redundancy in 28-112 layers, even with a batch size of 1. FB_base also cannot handle Reddit with 112 layers, but GraNNDis switches to cooperative batching with expansion-aware sampling, which becomes the only available framework.

8.6 Analysis of Memory Consumption

Figure 13a shows the memory consumption of the baseline MB_base (with SAGE) and cooperative batching (CoB-SAGE, CoB-EAS) in a deep GNN of 56 layers and 256 hidden dimensions with the same batch size. Cooperative batching significantly reduces the redundant memory usage of MB_base, and EAS additionally suppresses it. Figure 13b gives further insights about the memory usage of the baselines and GraNNDis. We generated random synthetic graphs of various numbers of vertices, following the average degree of the Reddit dataset. We used the 28-layer model, and colored $n|V|$ indicates the maximum batch size. GRD consumes slightly more memory than FB_base from flexible preloading. When the graph becomes larger, FB_base fails to handle it due to memory limit. In contrast, GRD keeps training by transitioning to mini-batch training with cooperative batching, which is the advantage of a unified framework. On the other hand, MB_base consumes significantly larger memory than GRD with the same batch size due to redundant memory usage. Due to the neighbor explosion, MB_base fails to train large graphs over $4|V|$. However, GRD can train them through cooperative batching and expansion-aware sampling.

8.7 Full-Batch Sampling/Staleness Methods

Some prior works have introduced sampling- [76] or staleness- [56, 77] based methods to address the massive communication overhead of full-batch training. Sancus [56] (SCS) uses the staled historical embeddings. PipeGCN [77] (PIPE) overlaps the communication by computation through pipelining and uses the staled embeddings. BNS-GCN [76] (BNS) samples the communication to be around 10% and uses stale activations. Even though GraNNDis does not involve any staleness, we directly compared flexible preloading with expansion-aware sampling (FLX-EAS) to the methods described above. We used the Products and Reddit dataset using the open-sourced code of [56, 76, 77] in Figure 14. We used 3 layers containing 2 GCN layers and a single FC layer with a hidden size of 256. We fixed a learning rate of 0.01 and a dropout of 0.0 following [56].

As shown in Figure 14, the communication volume of broadcast-based SCS is large because it is proportional to the square of the
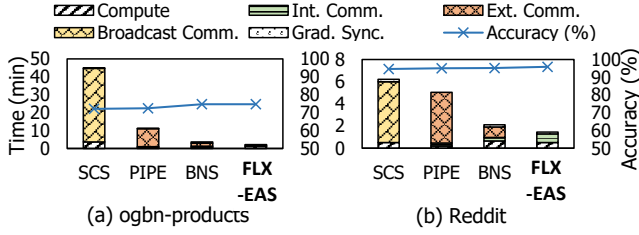
Figure 14: Training time breakdown and accuracy of GraNNDis with expansion-aware sampling compared to sampling- and staleness-based full-batch training methods.

total number of vertices. By contrast, the communication volume of all-to-all-based methods (PIPE, BNS, and FLX-EAS) is proportional to the total number of edges. Therefore, those works provide more speedup over SCS on the Product dataset, which has a lower graph density (#Edges/#Vertices) than the Reddit dataset. GraNNDis (FLX-EAS) shows sufficient speedup over all prior works, with values ranging from 1.45× to 28.04× because they suffer from the slow inter-server communication. Further, GraNNDis provides the same or more stable accuracy than the baselines.

8.8 Accuracy/Actual Total Error Comparison

We compared the accuracy of non-sampled FB_base and expansion-aware sampling (FLX-EAS) with 3-layer GraphSAGE model in Table 4. GraNNDis provides comparable accuracy to the non-sampled training by keeping the intra-server dependency and due to the effect of a large batch, as discussed in Section 5.3.2.

To have a deeper insight, we also directly measured the actual total error coming from expansion-aware sampling, fanout sampling, and subgraph sampling in Figure 15a by measuring the total value of Equation (8) in Section 5.3.2 using the trained weights. Due to large β (i.e., the portion of boundary vertices), subgraph sampling shows substantially larger total errors than others. Expansion-aware sampling obtains the advantage of the large batch (small β) and provides low actual errors. Fanout sampling also shows low errors but suffers from increased errors in a high average degree dataset (i.e., Reddit). This is because fanout sampling restricts fixed fanout for all vertices. For Arxiv, fanout sampling shows slightly lower errors than expansion-aware sampling, but it is natural because the average degree of Arxiv is around 7, much smaller than the used fanout 25. The sensitivity of β on batch sizes ranges from 0.08 to 0.56, as shown in Figure 15b. β is smaller than 0.6, even using 16 servers ($|V|/16$), so expansion-aware sampling can provide stable convergence in more extensive cluster settings.

9 DISCUSSION

9.1 Quantitative Memory Analysis

We approximately formulated GraNNDis's additional memory usage as follows. When scaling the number of servers, the total aggregated additional memory usage increases, but the additional memory usage per GPU is stable due to the effect of utilizing multiple servers/GPUs and expansion-aware sampling.

For formulation, we set some notations and assumptions as below. We use two number of servers (N_{s_1} and N_{s_2}) and $N_{s_2} = m \times N_{s_1}$. We also assume that we already applied expansion-aware sampling

Table 4: Test Accuracy of FB_base and FLX-EAS

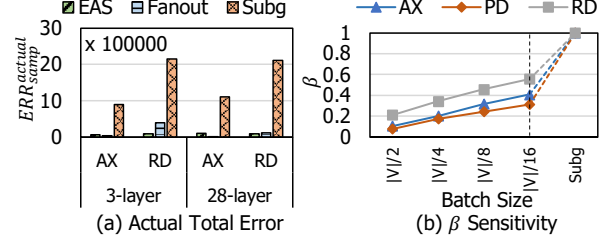| Dataset	Arxiv	Products	Reddit	IGB (S)	IGB (M)	Papers
FB_base	69.03%	75.69%	96.33%	74.63%	73.49%	OOM
FLX-EAS	69.21%	75.31%	96.33%	74.70%	73.55%	60.84%

Figure 15: (a) Actual values of total errors from samplings on the four-server setting and (b) sensitivity of β on batch sizes.

of $h = 1$ (1-hop) for brevity. H indicates the hidden size, and we set the initial feature size as the same as the hidden size. $1 < \alpha_1 < \alpha_2$ are graph- and #servers-dependent per-layer expansion factor. α_2 is larger than α_1 because a graph expands more when #target vertices is small. $\alpha_2 = n \times \alpha_1$ where $n << m$ due to the min-cut partitioning algorithm (i.e., METIS). The dependency graph size grows α for 1-hop because we applied expansion-aware sampling.

The memory usage/GPU of flexible preloading using expansion-aware sampling ($h = 1$) with s_1 servers ($M_{preload}^{s_1}$) is formulated as below ($2\times$ for forward/backward):

$$M_{preload}^{s_1} = 2 \times \frac{VHL\alpha_1}{N_{s_1}N_g}. \tag{12}$$

Now, we further formulate how the additional memory usage changes when increasing the number of servers to s_2. When we compare the total aggregated memory usage of the whole server clusters, this can be formulated as follows:

$$2 \times VHL(\alpha_2 - \alpha_1) > 0 \ (\because \alpha_1 < \alpha_2). \tag{13}$$

This indicates that the total aggregated memory usage increases with larger #servers. However, in terms of each GPU, we can compare the additional memory usage per GPU with s_1 and s_2 servers ($M_{preload}^{s_2}/M_{preload}^{s_1}$) as follows:

$$\frac{M_{preload}^{s_2}}{M_{preload}^{s_1}} = \frac{N_{s_1}\alpha_2}{N_{s_2}\alpha_1} = \frac{n}{m} < 1. \tag{14}$$

When using the min-cut partitioning and expansion-aware sampling, $n << m$ usually holds. Therefore, the additional memory usage per GPU is stable when scaling the number of servers.

9.2 Application on High-End Systems

While we tested GraNNDis on a four-server cluster with fast Infiniband (32Gbps) and NVLink Bridges (112GB/s), recent high-end systems (e.g., DGX-A100) often introduce much faster inter-server connection (e.g., 200Gbps Infiniband) and intra-server connection (e.g., 600GB/s NVLink Switch). We believe GraNNDis would provide speedup in such systems because their intra-/interbandwidth ratio (600GB/200Gb) does not significantly differ from ours (112GB/32Gb). GraNNDis mainly benefits from trade-offing intra-server communication with inter-server communication, so the speedup will persist.

10 RELATED WORK

Large graphs and deep GNNs. There have been attempts to increase the layer of GNN, as in traditional CNN models [24, 25]. It was shown that naively increasing GNN layers suffer from oversmoothing issues. Therefore, [40, 41] applied residual connection and preactivation to deep GNNs to mitigate it. These deep GNNs have shown superior accuracy in large graph datasets [26, 69]. To meet memory limitations, many previous [6, 13, 23, 87] works have used a sampling method in GNN training. [31, 48] further tried to reduce the sampling overhead by overlapping the data transfer with computation. [9, 93] use the subgraph structure of graphs and find local neighbors. Some works also focused on the overhead of preparing data for training and developed a cache policy [45, 46, 87] or sampling [76] to reduce feature retrieving costs.

GNN frameworks. There are many recent works [11, 45, 47, 56, 71, 72, 96, 97, 99] which aim to perform distributed training in a large graph. Most frameworks focus on how to handle the dependencies. [99] separately stores attributes of graphs in different workers and caches frequent neighbors. [94] designed k-hop neighborhood for each node and therefore enables parameter-server [42] based training. However, [94, 99] only support CPU-based GNN training. [96, 97] attempt to handle the dependency among vertices with caching and, therefore, suffer from the redundant computation. On the other hand, [11, 28, 56, 71, 72, 77] attempt to solve this dependency with communication. [72] uses sequential broadcast methods to handle the vertex dependency among distributed workers. [56] further accelerates it by using historical embeddings to avoid communication. [71] considers low-end servers, but it is limited to CPU servers. While focusing on the limitation of each approach, a hybrid approach was proposed [80], but with an unrealistic assumption that each server has only one GPU. [47] combined graph computation optimization with data partitioning and scheduling, and [11] adopted hypergraph partitioning model to derive optimized communication operations. However, the prior works [45, 47, 87] have not carefully considered a multi-server and multi-GPU environment, which has a large gap between internal and inter-server bandwidth. To our knowledge, this is the first approach to consider a multi-server cluster environment. Multi-GPU GNN training frameworks [1, 4, 65, 68, 79] are also widely used in GNN training. However, they mainly discuss based on the multi-GPU concept while not considering the multi-server setup. GraNNDis provides novel strategies considering the intra-/inter-server link bandwidth difference, including a sampling strategy. The contributions of multi-GPU GNN training could be applied to GraNNDis in terms of intra-server multi-GPUs.

Caching in GNN training. Many mini-batch training frameworks [7, 45, 68, 86] propose caching strategies for GNN training. They mainly focus on caching the initial features of mini-batches [45, 68], and introduce prefetch or dedicated caching algorithms [7, 45]. [86] provides shared GPU memory with multiple GPUs to use GPU memory as a cache. However, these works still face the redundancy of mini-batch because they do not consider the characteristics of multi-server clusters. In other words, as they target an individual GPU, they suffer from redundancy issues and need to cache duplicate initial features. On the other hand, GraNNDis proposes a kind of server-level caching through fast intra-server

communication, which is one of the main characteristics of recent clusters. As a result, GraNNDis significantly suppresses the redundancy and increases the scalability and throughput.

GNN training on limited environments. While GraNNDis targets a multi-server cluster with GNN training, many researches [54, 66, 75, 79, 95] suggest GNN training for limited environments. As recent graphs have become extremely large, those works provide methods to address the sampling bottleneck of mini-batch training. [54, 66, 75] mainly focus on efficiently utilizing SSDs when generating mini-batch subgraphs from SSDs. GraNNDis supports both full-/mini-batch training, so these works could be orthogonally applied when training is conducted with much larger graphs.

Graph processing and GNN inference. While GraNNDis mainly discusses GNN training, there exists a lot of graph processing [22, 36, 62] and GNN inference [21, 83, 89–91] research targeting various environments. [22] proposes an efficient graph accelerator structure for graph processing and [62] utilizes in-memory scatter-gather for addressing the random access of graph processing for acceleration. [36] emphasizes the importance of near-storage processing for graph processing with a dedicated accelerator. For GNN inference, many works [21, 83] propose various types of accelerators to reduce the inference latency. [89, 90] focus on efficiently utilizing on-chip memory and [91] exploits the sparsity of GNN activations while inferencing.

11 CONCLUSION

To the best of our knowledge, GraNNDis is the first distributed GNN training framework to fully consider the characteristics of multi-server clusters. By utilizing the chance provided by multi-server clusters, GraNNDis successfully addresses the inter-server communication bottleneck of distributed training. Additionally, GraNNDis mitigates the redundancy issue of mini-batch training by aggregating multiple GPUs in a server with fast intra-server links. Lastly, GraNNDis proposes a novel sampling method, suited for multi-server clusters, which significantly suppresses the neighbor explosion. As a result, GraNNDis is a fast GNN training framework that provides state-of-the-art training throughput.

DATA AVAILABILITY STATEMENT

The artifact of GraNNDis is available at [64]. The most recent version is kept updated at https://github.com/AIS-SNU/GraNNDis_Artifact. The artifact contains some GraNNDis code and evaluation runners for distributed environments. Also, it includes an environment setup script. For the details of the artifact and the expected results, please refer to Appendix A.

Jaeyong Song, Hongsun Jang, Hunseong Lim, Jaewon Jung, Youngsok Kim, and Jinho Lee

A ARTIFACT APPENDIX

A.1 Abstract

We provide GraNNDis's source code and additional code for setup and execution. For the most recent version of GraNNDis's description, please refer to the up-to-date artifact link in GitHub [64].

A.2 Artifact Summary

- **Algorithm:** Distributed graph neural network training.
- **Program:** Python with PyTorch and DGL.
- **Dataset:** Sample graph datasets (ogbn-arxiv/products [26], Reddit [84])
- **Run-time environment:** Ubuntu 22.04 or higher.
- **Hardware:** Multiple servers and each server has multiple GPUs. The internal server bandwidth (e.g., NVLink) is recommended to be much higher than the external server bandwidth (e.g., Ethernet).
- **Metrics:** Execution time and accuracy.
- **Recommended disk/memory space:** 1TB/256GB.
- **Preparation time:** For software setting, it takes less than an hour.
- **Experiment time:** It takes less than two hours for sample datasets.
- **Code licenses:** MIT license.
- **Archived DOI:** https://doi.org/10.5281/zenodo.12738844
- **Up-to-date artifact:** https://github.com/AIS-SNU/GraNNDis_Artifact

A.3 Description

A.3.1 How to access. Please access the artifact through the archived DOI [64] or the up-to-date artifact link.

A.3.2 Hardware dependencies. Muti-server environment, and each server is equipped with multiple GPUs. Internal server interconnect (e.g., NVLink) is much faster than external server interconnect (e.g., 10G Ethernet).

A.3.3 Software dependencies. CUDA/CuDNN 11.8 Setting (Make sure to include CUDA paths). Anaconda Setting. The NFS environment has more than two servers, and each server has multiple GPUs. Servers must be accessible by SSH connection without password using ssh-copy-id (e.g., ssh [user]@[server]).

A.3.4 Datasets. For the artifact evaluation, we use three sample datasets (Arxiv, Reddit, and Products), which are widely used and easily accessible. In the following subsection, the datasets will be automatically downloaded.

A.4 Installation

A.4.1 Software installation. Before installation, pre-install the Anaconda environment manager. Then execute the following:

```
$ conda update -y conda
$ conda create -n granndis_ae python=3.10 -y
$ conda activate granndis_ae
$ conda install -c conda-forge -c pytorch -c nvidia \
  -c dglteam/label/th21_cu118 \
  --file conda-requirements.txt -y
$ pip install -r pip-requirements.txt
```

A.4.2 Dataset preparation.

```
$ cd Codes
$ chmod +x brief_masking_test.sh
$ ./brief_masking_test.sh
```

While running the script, you may be required to type y to download a dataset. The logs will be saved in `Codes/masking_test_logs/` if the tests are successfully conducted.

A.5 Experiment workflow

Some users are unfamiliar with the distributed training procedure, so we provide simple distributed experiment launchers at `AE/*.py`. Before reproducing, users must change the configuration fields in the config file (`AE/configs.py`).

```
global_configs = {
    'env_loc': '(...)/envs/granndis_ae/bin/python',
    'runner_loc': '(...)/Codes/main.py',
    'our_runner_loc': '(...)/Codes/our_main.py',
    'workspace_loc': '(...)/GraNNDis_Artifact/',
    'data_loc': '~/datasets/granndis_ae/',
    'num_runners': 2,
    'gpus_per_server': 4,
    'hosts': ['192.168.0.5', '192.168.0.6']
}
```

After modification, the following commands will show the artifact evaluation results.

```
$ sh run_ae.sh # run AE scripts
$ sh parse_ae.sh # parse AE results
```

A.6 Evaluation and expected results

The results will be saved in the `AE*_results.log`. All FLX, CoB (with SAGE sampling), and EAS would generally show significant speedup over the baseline optimized full-batch training because GraNNDis minimizes the slow external server communication (Appendix A.6.1). EAS (FLX-EAS) is expected to show more speedup than FLX, especially in larger datasets, such as Products. EAS usually shows higher speedup than CoB (especially in larger datasets) while providing comparable accuracy, as shown in Appendix A.6.2 (accuracy result). Please note that the result can fluctuate when the inter-server connection is shared with the cluster's NFS file system. In this case, running multiple trials will show the trend mentioned above. The following are the example results of running the above procedure on the authors' remote machine.

A.6.1 Throughput (execution time).

```
+------------------------------------------------------+
|             Throughput Results for Arxiv             |
+--------+-----------------+-----------------+---------+
| Method | Total Time (sec) | Comm Time (sec) | Speedup |
+--------+-----------------+-----------------+---------+
| Opt_FB |      15.40      |       9.85      |   1.00  |
|  FLX   |       8.60      |       2.37      |   1.79  |
|  CoB   |       8.78      |       2.58      |   1.75  |
|  EAS   |      11.67      |       3.70      |   1.32  |
+--------+-----------------+-----------------+---------+

+------------------------------------------------------+
|             Throughput Results for Reddit            |
+--------+-----------------+-----------------+---------+
| Method | Total Time (sec) | Comm Time (sec) | Speedup |
+--------+-----------------+-----------------+---------+
| Opt_FB |     449.27      |     422.35      |   1.00  |
|  FLX   |      87.55      |      49.67      |   5.13  |
|  CoB   |      90.44      |      49.98      |   4.97  |
```

```
| EAS   |     75.16       |     40.34       |  5.98  |
+--------+-----------------+-----------------+--------+

+--------------------------------------------------------+
|             Throughput Results for Products            |
+--------+-----------------+-----------------+--------+
| Method | Total Time (sec) | Comm Time (sec) | Speedup |
+--------+-----------------+-----------------+--------+
| Opt_FB |     79.67       |     69.15       |  1.00  |
| FLX    |     20.03       |      6.36       |  3.98  |
| CoB    |     21.85       |      8.33       |  3.65  |
| EAS    |     18.23       |      5.78       |  4.37  |
+--------+-----------------+-----------------+--------+
```

The results show that the optimized full-batch training baseline (Opt_FB) suffers from communication overhead, while flexible preloading (FLX)/cooperative batching (CoB) addresses such an issue through server-wise preloading. Expansion-aware sampling (EAS) further accelerates the training through server boundary-aware sampling. This trend becomes vivid in larger datasets (Reddit and Products).

A.6.2   Accuracy.

```
+------------------------------------+
| Accuracy Comparison (FB vs. FLX-EAS) |
+--------+-------+--------+------------+
| Method | Arxiv | Reddit |  Products  |
+--------+-------+--------+------------+
|   FB   | 0.69  |  0.96  |    0.76    |
|  EAS   | 0.69  |  0.96  |    0.76    |
+--------+-------+--------+------------+
```

As expansion-aware sampling (EAS) only targets sample server boundary vertices, contributing to acceleration, it achieves comparable accuracy to the original full-batch training.

A.7   Notes

For detailed arguments and distributed launch processes, please refer to the additional guidelines of the readme in the artifact.

REFERENCES

[1] Xin Ai, Qiange Wang, Chunyu Cao, Yanfeng Zhang, Chaoyi Chen, Hao Yuan, Yu Gu, and Ge Yu. 2024. NeutronOrch: Rethinking Sample-Based GNN Training under CPU-GPU Heterogeneous Environments. *Proceedings of the VLDB Endowment (VLDB)* 17, 8 (may 2024), 1995–2008.

[2] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P. Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E. Smidt, and Boris Kozinsky. 2022. E(3)-Equivariant Graph Neural Networks for Data-Efficient and Accurate Interatomic Potentials. *Nature Communications* 13, 1 (2022), 2453.

[3] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*.

[4] Zhenkun Cai, Qihui Zhou, Xiao Yan, Da Zheng, Xiang Song, Chenguang Zheng, James Cheng, and George Karypis. 2023. DSP: Efficient GNN Training with Multiple GPUs. In *ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*.

[5] Hanqiu Chen and Cong Hao. 2023. DGNN-Booster: A Generic FPGA Accelerator Framework For Dynamic Graph Neural Network Inference. In *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[6] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations (ICLR)*.

[7] Weijian Chen, Shuibing He, Yaowen Xu, Xuechen Zhang, Siling Yang, Shuang Hu, Xian-He Sun, and Gang Chen. 2023. iCache: An Importance-Sampling-Informed Cache for Accelerating I/O-Bound DNN Model Training. In *International Symposium on High-Performance Computer Architecture (HPCA)*. 220–232.

[8] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. 2019. Multi-label image recognition with graph convolutional networks. In *IEEE/CVF conference on computer vision and pattern recognition (CVPR)*.

[9] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[11] Gunduz Vehbi Demirci, Aparajita Haldar, and Hakan Ferhatosmanoglu. 2022. Scalable Graph Convolutional Network Training on Distributed-Memory Systems. *Proceedings of the VLDB Endowment (VLDB)* (2022).

[12] Ailin Deng and Bryan Hooi. 2021. Graph Neural Network-Based Anomaly Detection in Multivariate Time Series. In *AAAI conference on artificial intelligence (AAAI)*.

[13] Jialin Dong, Da Zheng, Lin F. Yang, and George Karypis. 2021. Global Neighbor Sampling for Mixed CPU-GPU Training on Giant Graphs. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[14] Facebook 2023. GLOO. https://github.com/facebookincubator/gloo, visited on 2023-02-01.

[15] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph Neural Networks for Social Recommendation. In *The World Wide Web Conference (WWW)*.

[16] Matthias Fey and Jan Eric Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. *ICLR Workshop on Representation Learning on Graphs and Manifolds (ICLRW)* (2019).

[17] M. Fey, J. E. Lenssen, F. Weichert, and J. Leskovec. 2021. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In *International Conference on Machine Learning (ICML)*.

[18] Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. 2017. Protein Interface Prediction using Graph Convolutional Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[19] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P3: Distributed Deep Graph Learning at Scale.. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[20] Johannes Gasteiger, Florian Becker, and Stephan Günnemann. 2021. GemNet: Universal Directional Graph Neural Networks for Molecules. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[21] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, and Martin C. Herbordt. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *International Symposium on Microarchitecture (MICRO)*.

[22] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. 2016. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *International Symposium on Microarchitecture (MICRO)*.

[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE/CVF conference on computer vision and pattern recognition (CVPR)*.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European Conference on Computer Vision (ECCV)*.

[26] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[27] Binxuan Huang and Kathleen Carley. 2019. Syntax-Aware Aspect Level Sentiment Classification with Graph Attention Networks. In *Conference on Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

[28] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Graph Neural Networks with Roc. In *Conference on Machine Learning and Systems (MLSys)*.

[29] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (High Bandwidth Memory) DRAM Technology and Architecture. In *IEEE International Memory Workshop (IMW)*.

[30] Tim Kaler, Alexandros-Stavros Iliopoulos, Philip Murzynowski, Tao B. Schardl, Charles E. Leiserson, and Jie Chen. 2023. Communication-Efficient Graph Neural Networks with Probabilistic Neighborhood Expansion Analysis and Caching. In *Conference on Machine Learning and Systems (MLSys)*.

[31] Tim Kaler, Nickolas Stathas, Anne Ouyang, Alexandros-Stavros Iliopoulos, Tao Schardl, Charles E Leiserson, and Jie Chen. 2022. Accelerating training and inference of graph neural networks with fast sampling and pipelining. In *Conference on Machine Learning and Systems (MLSys)*.

[32] George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing* (1998).

[33] Arpandeep Khatua, Vikram Sharma Mailthody, Bhagyashree Taleka, Tengfei Ma, Xiang Song, and Wen-mei Hwu. 2023. IGB: Addressing The Gaps In Labeling, Features, Heterogeneity, and Size of Public Graph Datasets for Deep Learning Research. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[34] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[35] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* (1998).

[36] Jinho Lee, Heesu Kim, Sungjoo Yoo, Kiyoung Choi, H. Peter Hofstee, Gi-Joon Nam, Mark R. Nutter, and Damir Jamsek. 2017. ExtraV: boosting graph processing near storage with a coherent accelerator. *Proceedings of the VLDB Endowment (VLDB)* 10, 12 (aug 2017), 1706–1717.

[37] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[38] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. 2020. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems (IEEE TPDS)* (2020).

[39] Guohao Li, Matthias Müller, Bernard Ghanem, and Vladlen Koltun. 2021. Training Graph Neural Networks with 1000 Layers. In *International Conference on Machine Learning (ICML)*.

[40] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. DeepGCNs: Can GCNs Go As Deep As CNNs?. In *International Conference on Computer Vision (ICCV)*.

[41] Guohao Li, Chenxin Xiong, Ali Thabet, and Bernard Ghanem. 2020. Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739* (2020).

[42] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[43] Yi-Lun Liao and Tess Smidt. 2023. Equiformer: Equivariant Graph Attention Transformer for 3D Atomistic Graphs. In *International Conference on Learning Representations (ICLR)*.

[44] Yi-Lun Liao, Brandon Wood, Abhishek Das*, and Tess Smidt*. 2024. EquiformerV2: Improved Equivariant Transformer for Scaling to Higher-Degree Representations. In *International Conference on Learning Representations (ICLR)*.

[45] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. 2020. PaGraph: Scaling GNN Training on Large Graphs via Computation-Aware Caching. In *ACM Symposium on Cloud Computing (SoCC)*.

[46] Tianfeng Liu, Yangrui Chen, Dan Li, Chuan Wu, Yibo Zhu, Jun He, Yanghua Peng, Hongzheng Chen, Hongzhi Chen, and Chuanxiong Guo. 2021. BGL: GPU-Efficient GNN Training by Optimizing Graph Data I/O and Preprocessing. *arXiv preprint arXiv:2112.08541* (2021).

[47] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. NeuGraph: Parallel Deep Neural Network Computation on Large Graphs.. In *USENIX Annual Technical Conference (ATC)*.

[48] Seung Won Min, Kun Wu, Sitao Huang, Mert Hidayetoğlu, Jinjun Xiong, Eiman Ebrahimi, Deming Chen, and Wen-mei Hwu. 2021. Large Graph Convolutional Network Training with GPU-Oriented Data Communication Architecture. *Proceedings of the VLDB Endowment (VLDB)* (2021).

[49] Zhewei Wei Ming Chen, Bolin Ding Zengfeng Huang, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *International Conference on Machine Learning (ICML)*.

[50] NVIDIA. 2023. InfiniBand Network. https://docs.nvidia.com/networking/display/MLNXOFEDv493150/InfiniBand+Network,visited on 2023-01-30.

[51] NVIDIA. 2023. NCCL. https://github.com/NVIDIA/nccl,visited on 2023-02-01.

[52] NVIDIA. 2023. NVLink Bridge. https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/,visited on 2023-06-01.

[53] Kenta Oono and Taiji Suzuki. 2020. Graph Neural Networks Exponentially Lose Expressive Power for Node Classification. In *International Conference on Learning Representations (ICLR)*.

[54] Yeonhong Park, Sunhong Min, and Jae W. Lee. 2022. Ginex: SSD-enabled billion-scale graph neural network training on a single machine via provably optimal in-memory caching. *Proceedings of the VLDB Endowment (VLDB)* 15, 11 (jul 2022), 2626–2639.

[55] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. (2017).

[56] Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. 2022. Sancus: Staleness-Aware Communication-Avoiding Full-Graph Decentralized Training in Large-Scale Graph Neural Networks. *Proceedings of the VLDB Endowment (VLDB)* (2022).

[57] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou Huang. 2020. Self-Supervised Graph Transformer on Large-Scale Molecular Data. *Advances in Neural Information Processing Systems (NeurIPS)* (2020).

[58] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations (ICLR)*.

[59] Rishov Sarkar, Stefan Abi-Karam, Yuqi He, Lakshmi Sathidevi, and Cong Hao. 2023. FlowGNN: A Dataflow Architecture for Real-Time Workload-Agnostic Graph Neural Network Inference. In *International Symposium on High-Performance Computer Architecture (HPCA)*.

[60] Marco Serafini and Hui Guan. 2021. Scalable Graph Neural Network Training: The Case for Sampling. *SIGOPS Oper. Syst. Rev.* (2021).

[61] Zhihao Shi, Xize Liang, and Jie Wang. 2023. LMC: Fast Training of GNNs via Subgraph Sampling with Provable Convergence. In *International Conference on Learning Representations (ICLR)*.

[62] Changmin Shin, Taehee Kwon, Jaeyong Song, Jae Hyung Ju, Frank Liu, Yeonkyu Choi, and Jinho Lee. 2024. A Case for In-Memory Random Scatter-Gather for Fast Graph Processing. *IEEE Computer Architecture Letter (IEEE CAL)* (2024).

[63] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. 2020. Graph neural networks in particle physics. *Machine Learning: Science and Technology* (2020).

[64] Jaeyong Song, Hongsun Jang, Hunseong Lim, Jaewon Jung, Youngsok Kim, and Jinho Lee. 2024. AIS-SNU/GraNNDis_Artifact: Artifact Evaluation Submission. https://doi.org/10.5281/zenodo.12738844

[65] Jie Sun, Li Su, Zuocheng Shi, Wenting Shen, Zeke Wang, Lei Wang, Jie Zhang, Yong Li, Wenyuan Yu, Jingren Zhou, and Fei Wu. 2023. Legion: Automatically Pushing the Envelope of Multi-GPU System for Billion-Scale GNN Training. In *USENIX Annual Technical Conference (USENIX ATC 23)*. 165–179.

[66] Jie Sun, Mo Sun, Zheng Zhang, Jun Xie, Zuocheng Shi, Zihan Yang, Jie Zhang, Fei Wu, and Zeke Wang. 2023. Helios: An Efficient Out-of-core GNN Training System on Terabyte-scale Graphs with In-memory Performance. *arXiv preprint arXiv:2310.00837* (2023).

[67] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[68] Zeyuan Tan, Xiulong Yuan, Congjie He, Man-Kit Sit, Guo Li, Xiaoze Liu, Baole Ai, Kai Zeng, Peter Pietzuch, and Luo Mai. 2023. Quiver: Supporting GPUs for Low-Latency, High-Throughput GNN Serving with Workload Awareness. *arXiv preprint* (2023).

[69] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnet-Miner: Extraction and Mining of Academic Social Networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[70] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *IJHPCA* (2005), 49–66.

[71] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[72] Alok Tripathy, Katherine Yelick, and Aydın Buluç. 2020. Reducing Communication in Graph Neural Network Training. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*.

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[74] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*.

[75] Roger Waleffe, Jason Mohoney, Theodoros Rekatsinas, and Shivaram Venkataraman. 2023. MariusGNN: Resource-Efficient Out-of-Core Training of Graph Neural Networks. In *European Conference on Computer Systems (EuroSys)*.

[76] Cheng Wan, Youjie Li, Ang Li, Nam Sung Kim, and Yingyan Lin. 2022. BNS-GCN: Efficient Full-Graph Training of Graph Convolutional Networks with Partition-Parallelism and Random Boundary Node Sampling. In *Conference on Machine Learning and Systems (MLSys)*.

[77] Cheng Wan, Youjie Li, Cameron R. Wolfe, Anastasios Kyrillidis, Nam Sung Kim, and Yingyan Lin. 2022. PipeGCN: Efficient Full-Graph Training of Graph Convolutional Networks with Pipelined Feature Communication. In *International Conference on Learning Representations (ICLR)*.

[78] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. 2019. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315* (2019).

[79] Qiange Wang, Yao Chen, Weng-Fai Wong, and Bingsheng He. 2023. HongTu: Scalable Full-Graph GNN Training on Multiple GPUs. *Proceedings of the ACM on Management of Data* 1, 4, Article 246 (2023).

[80] Qiange Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen, Xiaodong Zhang, and Ge Yu. 2022. NeutronStar: Distributed GNN Training with Hybrid Dependency Management. In *International Conference on Management of Data (SIGMOD)*.

[81] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. 2021. Representing Long-Range Context for Graph Neural Networks with Global Attention. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[82] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations (ICLR)*.

[83] Mingyu Yan, Lei Deng, Xing Hu, Ling Liang, Yujing Feng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, and Yuan Xie. 2020. HyGCN: A gcn accelerator with hybrid architecture. In *International Symposium on High-Performance Computer Architecture (HPCA)*.

[84] Pinar Yanardag and SVN Vishwanathan. 2015. Deep graph kernels. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[85] Chaoqi Yang, Ruijie Wang, Shuochao Yao, Shengzhong Liu, and Tarek Abdelzaher. 2020. Revisiting over-smoothing in deep GCNs. *arXiv preprint* (2020).

[86] Dongxu Yang, Junhong Liu, Jiaxing Qi, and Junjie Lai. 2022. WholeGraph: A Fast Graph Neural Network Training Framework with Multi-GPU Distributed Shared Memory Architecture. In *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*.

[87] Jianbang Yang, Dahai Tang, Xiaoniu Song, Lei Wang, Qiang Yin, Rong Chen, Wenyuan Yu, and Jingren Zhou. 2022. GNNLab: A Factored System for Sample-Based GNN Training over GPUs. In *European Conference on Computer Systems (EuroSys)*.

[88] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do Transformers Really Perform Badly for Graph Representation?. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[89] Mingi Yoo, Jaeyong Song, Hyeyoon Lee, Jounghoo Lee, Namhyung Kim, Youngsok Kim, and Jinho Lee. 2022. Slice-and-Forge: Making Better Use of Caches for Graph Convolutional Network Accelerators. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

[90] Mingi Yoo, Jaeyong Song, Jounghoo Lee, Namhyung Kim, Youngsok Kim, and Jinho Lee. 2021. Making a Better Use of Caches for GCN Accelerators with Feature Slicing and Automatic Tile Morphing. *IEEE Computer Architecture Letter (IEEE CAL)* (2021).

[91] M. Yoo, J. Song, J. Lee, N. Kim, Y. Kim, and J. Lee. 2023. SGCN: Exploiting Compressed-Sparse Features in Deep Graph Convolutional Network Accelerators. In *International Symposium on High-Performance Computer Architecture (HPCA)*.

[92] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the Depth and Scope of Graph Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (Eds.).

[93] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *International Conference on Learning Representations (ICLR)*.

[94] Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. 2020. AGL: A Scalable System for Industrial-Purpose Graph Machine Learning. *Proceedings of the VLDB Endowment (VLDB)* (2020).

Jaeyong Song, Hongsun Jang, Hunseong Lim, Jaewon Jung, Youngsok Kim, and Jinho Lee

[95] Xin Zhang, Yanyan Shen, Yingxia Shao, and Lei Chen. 2023. DUCATI: A Dual-Cache Training System for Graph Neural Networks on Giant Graphs with the GPU. *Proceedings of the ACM on Management of Data* 1, 2, Article 166 (2023).

[96] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. *arXiv preprint arXiv:2010.05337* (2020).

[97] Da Zheng, Xiang Song, Chengru Yang, Dominique LaSalle, and George Karypis. 2022. Distributed Hybrid CPU and GPU Training for Graph Neural Networks on Billion-Scale Heterogeneous Graphs. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[98] Zhe Zhou, Cong Li, Xuechao Wei, Xiaoyang Wang, and Guangyu Sun. 2022. GNNear: Accelerating Full-Batch Training of Graph Neural Networks with Near-Memory Processing. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

[99] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: A Comprehensive Graph Neural Network Platform. *Proceedings of the VLDB Endowment (VLDB)* (2019).