Scaling New Heights: Transformative Cross-GPU Sampling for Training Billion-Edge Graphs

Yaqi Xia
School of Computer Science
Wuhan University
yaqixia@whu.edu.cn

Donglin Yang
NVIDIA
dongliny@nvidia.com

Xiaobo Zhou
IOTSC
University of Macau
waynexzhou@um.edu.mo

Dazhao Cheng*
School of Computer Science
Wuhan University
dcheng@whu.edu.cn

***Abstract*—Efficient training of Graph Neural Networks (GNNs) on billion-edge graphs poses significant challenges due to memory constraints and data transfer bottlenecks, particularly affecting GPU-based sampling. Traditional methods either face severe CPU-GPU data transfer bottlenecks or encounter excessive data shuffling and synchronization overheads in multi-GPU setups.**

To overcome these challenges in GNN training on large-scale graphs, we introduce HyDRA, a pioneering framework that elevates mini-batch, sampling-based training. HyDRA innovates in multi-GPU memory sharing and multi-node feature retrieval, transforming cross-GPU sampling by seamlessly integrating sampling and data transfer into a single kernel operation. It develops a hybrid pointer-driven data placement technique to enhance neighbor retrieval efficiency, designs a targeted replication strategy for high-degree vertices to reduce communication overhead, and leverages dynamic cross-batch data orchestration with pipelining to minimize redundant data transfers. Evaluated on systems equipped with up to 64 A100 GPUs, HyDRA significantly outperforms current leading methods, achieving 1.4x to 5.3x faster training speeds compared to DSP and DGL-UVA and demonstrating up to a 42x improvement in multi-GPU scalability. HyDRA sets a new benchmark for high-performance GNN training at large scales.

I. Introduction

GNNs have become essential for extracting insights from graph-structured data, spanning diverse applications from social network analytics [1], [2] to bioinformatics [3], [4]. Sampling-based GNN training have demonstrated superior performance in node classification [5] and link prediction tasks [6], offering advantages such as enhanced model accuracy [7], [8], reduced computational complexity [9], [10], and improved generalization capabilities [11], [12] compared to their full-batch counterparts [13], [14].

Sampling-based GNNs, also referred to mini-batch GNNs, consist of two critical preparatory stages for batch data training: (1) graph sampling, where a subset of the large original graph is selected to form mini-batch topologies; and (2) feature retrieval, which involves access to the attributes of these selected vertices. However, the rise of large graph datasets, particularly those at the billion-edge scale, presents formidable challenges, particularly to GPU memory constraints. For example, a graph dataset with one billion edges can easily exceed 60 GB, surpassing the maximum GPU memory capacity of a single high-end GPU available to us, which is the 40GB A100. This size discrepancy makes it impossible to store the entire graph, let alone allocate additional memory for model training and intermediate data.

Traditional sampling-based frameworks, such as DGL [15] and PyG [16], mainly rely on CPU memory to store graph data and perform topology sampling. The sampled features are constructed as the mini-batches which are subsequently transferred to the GPU via PCIe for model training. This process, however, is bottlenecked by the limited CPU-GPU transfer bandwidth and the slow CPU sampling speed, leading to underutilized GPUs. An attempt to address this, DGL-UVA [17], employs Unified Virtual Memory Address (UVA) technology for direct GPU access to graph topology, but the random nature of sampling and numerous small data transfers still strain the PCIe bandwidth, resulting in low throughput. More recent strategies, such as PaGraph [18], BGL [19], and GNNLab [20], adopt a hybrid data storage approach, caching essential data on the GPU. These methods prioritize different caching strategies, from favoring frequently accessed data to employing advanced predictions for cache management. Nonetheless, these solutions are still limited by the finite memory of a single GPU, and un-cached data must still endure the high costs of PCIe transfers.

DSP [7] stands as a pioneering framework in multi-GPU graph sampling, proposing a novel approach where the graph topology is divided among several GPUs within a single node. This approach is designed to facilitate cross-GPU neighbor retrieval through the use of collective communication techniques. Specifically, DSP employs a two-phase all-to-all communication strategy: the first phase involves distributing target vertices across GPUs based on their graph partitions, enabling each GPU to perform sampling independently. The second phase then reverses this process, gathering the distributed sampling results back to their original locations. Despite its innovations, DSP encounters two significant performance hurdles: 1) The randomness inherent in graph sampling necessitates extensive data shuffling, notably before and after all-to-all communications, which can account for over 60% of the sampling duration, as evidenced by our experiments (see Figure 3). 2) The reliance on multiple all-to-all communications introduces non-trivial cross-GPU synchronization issues, adversely impacting sampling efficiency.

In this paper, we address the identified limitations in current

*Corresponding author

SC24, November 17-22, 2024, Atlanta, Georgia, USA
979-8-3503-5291-7/24/$31.00 ©2024 IEEE

sampling-based GNN training methodologies, with a particular focus on billion-edge scale graphs. We introduce HyDRA, a novel framework designed to revolutionize scalable sampling-based GNN training across extensive graph structures. HyDRA introduces two key innovations: 1) Single-node multi-GPU memory sharing, which effectively eliminates the data shuffling and synchronization overheads typically associated with graph sampling processes; and 2) Multi-node feature retrieval, designed to minimize redundant data transfers and significantly enhance GPU utilization.

Firstly, we introduce a novel multi-GPU sampling method that leverages memory sharing across GPUs to significantly reduce the overheads commonly associated with the DSP framework. For graph topology data, which represents a relatively small fraction (approximately 10%) of the total graph data, we partition it across different GPUs within a single node to avoid the expensive inter-node communication. We merge the sampling and remote neighbor retrieval into a single kernel operation, thereby eliminating the extensive data shuffling and frequent cross-GPU synchronization required by the existing method. We further design a hybrid pointer-driven data management approach that facilitate sharing graph topology data across multiple GPUs while maintaining local mappings for the head and tail addresses of neighbor lists.

However, we encountered a challenge: integrating GPU memory sharing unexpectedly led to a dramatic increase in communication volume, in some instances by more than 8x, due to inefficient graph partitioning methods. Upon further investigation, we identified high-degree vertices, vertices with a large number of neighbors, as the primary culprits for this surge. To counter this, we develop a novel partitioning strategy based on the replication of high-degree vertices. By duplicating the neighbor lists of these vertices across each GPU partition, our method effectively converts what would be remote accesses into local reads. This innovation markedly diminishes communication overhead, thus addressing one of the critical bottlenecks in multi-GPU sampling.

Secondly, to optimize storage and computational efficiency, we strategically partition heavyweight feature data across multiple nodes and GPUs within the cluster. In the realm of multi-node feature retrieval, we aim to significantly accelerate the process through a hierarchical data reuse strategy, addressing the prevalent issue of excessive data movement. We note that in mini-batch training, particularly with larger batch sizes, there is a considerable overlap of data across consecutive batches. Traditionally, data from a previous batch is discarded and replaced by the new batch, ignoring potential efficiencies. To counter this waste, we introduce a dynamic cross-batch data orchestration mechanism designed to capture and repurpose this redundant data, thereby enhancing efficiency.

Expanding on this concept for multi-node, multi-GPU systems, where communication bandwidth can be highly variable, we adopt a layered approach to data reuse. This hierarchical method operates at three levels: within individual GPUs, across multiple GPUs in the same node, and across different nodes. By structuring data reuse to prioritize channels with higher bandwidth, our approach strategically minimizes communication delays. This systematic reuse not only leverages the inherent data locality and redundancy but also aligns data flow with network capabilities, resulting in a substantial reduction in overall communication overhead.

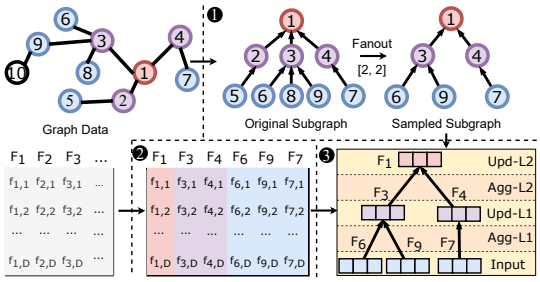


Fig. 1: Illustration of mini-batch training, separating graph sampling and feature retrieval. 'Agg-Ln' and 'Upd-Ln' denote the aggregation and update operations at layer n, respectively.

In summary, we highlight our contribution as follows:

1) Through analyzing the existing multi-GPU sampling strategy, we pinpoint a primary challenge in training billion-edge graphs: the distinction between coarse-grained neighbor retrieval and fine-grained sampling.
2) We design a transformative cross-GPU sampling approach that seamlessly merges sampling and data transfer into a single kernel operation. We further develop a hybrid pointer-driven data placement technique that significantly enhances neighbor retrieval efficiency.
3) We propose a high-degree vertex replication strategy that significantly reduces communication volumes caused by high-degree vertices, enhancing sampling efficiency by leveraging communication locality.
4) We develop a dynamic cross-batch data orchestration mechanism that optimizes intra-GPU data reuse and extends it to reduce cross-node communication in multi-node, multi-GPU settings.

HyDRA has been integrated into the PyTorch environment. Our comprehensive evaluation has shown its remarkable performance improvements. For single-node sampling efficiency, it achieves a remarkable 1.6x to 18.9x speedup over state-of-the-art methods DSP and DGL-UVA, along with a scalability speedup of 7.35x with 8 GPUs. HyDRA also enhances training efficiency to 1.4x to 5.3x, exhibiting a maximum speedup of 42x in multi-node setups with up to 64 A100 GPUs.

II. Background and Motivation

A. GNN Training Methodologies

1) Full-batch GNN Training: Full-batch training involves using the entire dataset for each iteration, updating weights based on all vertices and edges in the graph [13], [14]. However, challenges arise as graph sizes increase and due to the highly skewed degree distributions characteristic of real-world graphs [21], [22]. These factors contribute to poor scalability [18], [23] and reduced computational efficiency [24], thereby constraining its further development.

TABLE I: Breakdown of DGL-UVA and DSP performance.

Method	Data	Time (ms) / percentage(%) Sampling	Retrieving	Computation
DGL-UVA	Papers	28.45/33.47	43.82/51.56	12.72/14.97
	Twitter	34.04/21.87	104.24/60.98	17.34/11.14
	Products	42.17/54.38	27.21/35.09	8.17/10.54
DSP	Papers	12.11/31.93	13.33/35.14	12.49/32.93
	Twitter	16.15/27.13	27.40/46.03	15.98/26.84
	Products	18.85/48.01	11.99/30.54	8.42/21.45

2) Mini-batch GNN Training: Mini-batch training, as depicted in Figure 1, addresses the limitations of full-batch training by dividing the graph into smaller, more manageable mini-batches for iterative updates [8], [11], [25]–[29]. This method begins with batch sampling (❶), compiling subsets of graph vertices into mini-batches. It then proceeds to feature retrieval (❷), loading vertex features relevant to each mini-batch. Finally, it performs model computation (❸), applying GNN operations on the mini-batches through stacked neural network layers and sampled subgraphs.

In GNNs, the learning process at each layer primarily involves two operations: Aggregation and Update. For a given graph $G = (V, E)$, where V represents the vertices and E represents the edges, the operations for each vertex v at layer l are defined as follows:

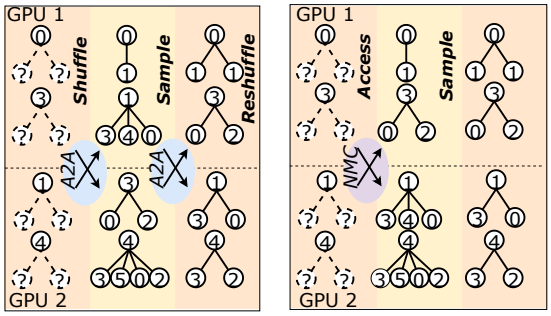
$$\mathrm{h}_v^{(l)} = \mathrm{UPDATE}^{(l)}\left(\mathrm{AGGREGATE}^{(l)}\left(\mathrm{h}_u^{(l-1)} : u \in \mathcal{N}(v)\right)\right) \quad (1)$$

Here, $\mathrm{h}_u^{(l-1)}$ represents the features of each neighbor u of vertex v at layer $(l-1)$, and $\mathcal{N}(v)$ signifies the set of neighbors of v. This formulation allows the model to effectively incorporate the graph's structure into the learning process.

B. Distributed Sampling-based GNN Training Bottlenecks

Unlike traditional deep neural networks, such as convolutional networks where computational focus is primarily on the model itself [30]–[35], GNNs face a unique bottleneck: the extensive demand of subgraph sampling and feature retrieval [19]. This challenge becomes even more pronounced in distributed training scenarios, where partitioning graph data incurs significant additional communication costs. Our empirical analysis, conducted on 4 A100 GPUs using various datasets, emphasizes the critical bottleneck present in two leading GNN frameworks, DSP [7] and DGL-UVA [17]. We found that sampling and feature retrieval alone constitute over 80% of the total training time, as elaborated in Table I. Such a substantial overhead significantly undermines both scalability and efficiency. Subsequently, we investigate the critical challenges associated with these processes, which remain inadequately addressed or neglected by existing research.

1) Challenge in Multi-GPU Sampling: Table I and Figure 2a show that DSP, a pioneering framework, significantly outperforms DGL-UVA in sampling efficiency by introducing multi-GPU sampling with efficient neighbor access via collective communication, highlighting the advantages of this approach. Despite efforts to reduce communication volume, DSP's utilization of coarse-grained collective communication



(a) Multi-GPU graph sampling in DSP. A2A is an abbreviation for all-to-all communication.

(b) Memory sharing sampling in HyDRA. NMC is an abbreviation for NVSHMEM communication.

Fig. 2: Two implementations of multi-GPU graph sampling.

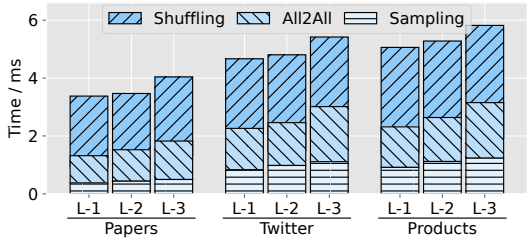


Fig. 3: Breakdown of the DSP multi-GPU sampling process, where L-N represents the N-th GNN layer

(i.e., all-to-all) leads to a distinction between it and fine-grained sampling, consequently encountering two primary challenges: (1) Data shuffling: graph sampling's inherent randomness necessitates intricate data organization. The necessary data preparation before and resetting after all-to-all communication leads to substantial data shuffling, significantly impacting efficiency. This issue is reflected in the DSP's performance breakdown in a 4 A100 GPU setup, as depicted in Figure 3, where shuffling consumes over 60% of the operation time. (2) Multiple all-to-all communication: DSP employs two primary rounds of all-to-all communication via NCCL [36], leading to considerable cross-GPU synchronization overheads.

To address these shortcomings, we want to propose a new solution based on GPU memory sharing, as demonstrated in Figure 2b. This approach facilitates multiple GPUs to collaboratively share and access graph topology information, allowing sampling and neighbor retrieval to be performed within the same kernel operation, thereby eradicating the repetitive data shuffling associated with data preparation and post-communication processes.

Recently, there has also been some work applying the idea of memory sharing to graph computing or GNN training. Existing research in graph computing, such as Ligra [37] and Congra [38], explores memory sharing for algorithms like BFS in multi-CPU setups. However, the focus of our work is on graph sampling in multi-GPU environments, which introduces unique access patterns and hardware character-

TABLE II: Communication volume of memory sharing implementation normalized to that of DSP implementation.

Method	Arxiv	Products	Papers
Range	0.42x	5.16x	1.96x
Random	0.66x	8.49x	1.12x
Metis	0.18x	4.96x	1.55x

istics, necessitating distinct design considerations. In GNN training, MGG [39] introduces this approach for full-graph training to achieve fine-grained computation-communication pipelining. Our work distinguishes from MGG in two key aspects: (1) MGG is optimized for full-batch training, whereas our approach utilizes mini-batch training based on sampling. Consequently, MGG does not encounter the bottlenecks associated with data shuffling and synchronization overheads during cross-GPU sampling, which are primary challenges for our method. Moreover, due to the randomness of the graph sampling process and the non-uniform distribution of graph topology, we must devise more specific graph partitioning schemes and intricate data placement strategies. (2) We target the scenario of training billion-edge scale graphs across multiple nodes and multiple GPUs, whereas MGG is designed for training relatively smaller graphs on a single node.

However, despite its advantages, adopting memory sharing introduces new challenges, particularly in balancing data transmission demands. To quantitatively measure the difference, we compare memory sharing to DSP with M GPUs and N target vertices. Their communication volumes are represented as:

$$C_{DSP} = N \times \frac{M-1}{M} \times (f+1) \tag{2}$$

$$C_{Memory\ Sharing} = N \times \frac{M-1}{M} \times d \tag{3}$$

where f is the fanout (the number of sampled vertices) and d represents the degree (number of neighbors) of a target vertex.

Therefore, in DSP implementation, each sampling operation involves transferring only the sampled neighbor vertices. In contrast, memory sharing implementation requires the transfer of all neighbors for a vertex. To compare the data transfer of memory sharing and DSP, we applied various graph partitioning methods, including range, random, and Metis [40], to distribute the graph across multiple GPUs. As shown in Table II, memory sharing communication volume significantly exceeded that of DSP for datasets other than Arxiv. The results present a crucial challenge: without addressing the increased communication volume, the high communication costs could substantially diminish the benefits of using memory sharing.

2) Challenge in Feature Retrieval: In most GNN training frameworks [15], [16], mini-batches operate independently, implying that the data processed in one iteration is promptly overwritten by the subsequent one. However, our observations reveal a critical insight: the sampling process often leads to an exponential increase in the number of neighbors, thereby substantially enlarging the proportion of feature data required for each subgraph. Consequently, there exists a notable overlap in the feature data required across successive batches.

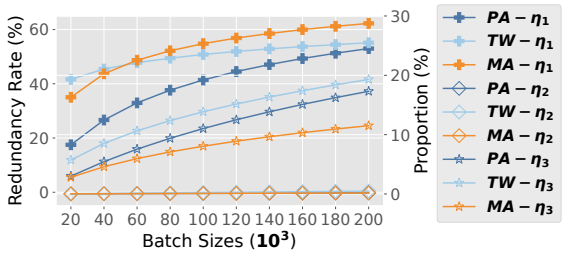
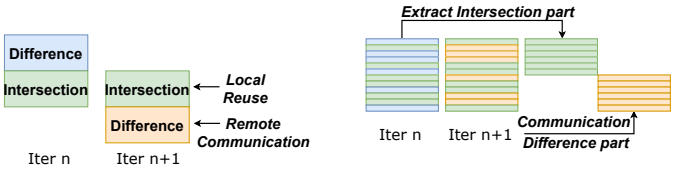


Fig. 4: Evaluation of redundancy rate and proportion of Papers (PA), MAG240M (MA), and Twitter (TW) datasets.



(a) Feature retrieval by local reuse & remote communication.

(b) The Feature data exhibits a random distribution in practice.

Fig. 5: Illustration of cross-batch feature reuse.

Specifically, we define the data overlap between consecutive batches as the redundancy rate, denoted by η_1, and quantify the proportion of data for target vertices and their neighbors in a mini-batch relative to the entire dataset as η_2 and η_3 respectively. We conducted experiments across various batch sizes and dataset combinations to observe the variations in η_1, η_2 and η_3, as depicted in Figure 4. The decision against smaller batch sizes is based on studies suggesting that larger batches may benefit model accuracy [41]–[43]. The results highlight two key observations: Despite the proportionally smaller data size for target vertices, the requisite feature data is significantly large, indicating substantial overhead in feature retrieving. Furthermore, there is a considerable overlap between consecutive batches, suggesting that many feature transmissions are redundant. This redundancy highlights potential optimizations in the feature retrieving process, underlining the importance of efficient data management to reduce unnecessary communication costs in distributed GNN training.

As shown in Figure 5a, we categorize the feature data required across two consecutive iterations (iter n and iter $n+1$) into two segments: the Intersection and the Difference. The Intersection represents the overlapping portion of feature data between iterations, which can be locally reused to decrease communication costs. Conversely, the Difference comprises the distinct elements that necessitate remote communication for access. In practice, the distribution of features across the Intersection and Difference segments is interleaved randomly, as illustrated in Figure 5b. Thus, it necessitates extracting the overlapping portion in one iteration and identifying, then exclusively communicating, the distinct portion in the subsequent iteration. Such complex data comparison and extraction tasks present significant challenges to end-to-end training efficiency and memory utilization, underscoring the need for innovative

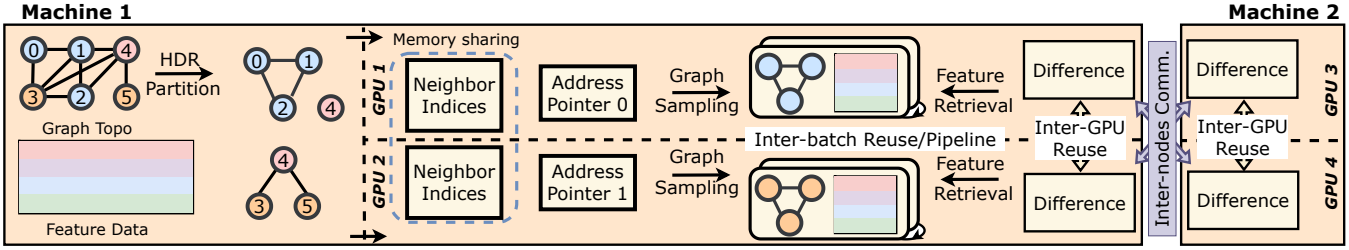


Fig. 6: The architecture of HyDRA, featuring memory sharing graph sampling and hierarchical cross-batch feature retrieval.

solutions to manage these intricacies effectively.

III. System Design of HyDRA

HyDRA, a scalable multi-node, multi-GPU GNN training framework, is designed to handle billion-edge graphs effectively. As depicted in Figure 6, HyDRA's system architecture introduces two primary innovations: (1) GPU Memory Sharing Based Sampling: This feature leverages hybrid pointer-driven data placement, high-degree vertex replication, and threshold pruning. Together, these techniques facilitate efficient neighbor information sharing and significantly reduce communication overhead. (2) Hierarchical Cross-Batch Feature Retrieval: By implementing dynamic cross-batch data orchestration alongside hierarchical data utilization and pipelining, this approach streamlines data transfer and cuts down on communication overhead, thereby boosting GPU utilization.

A. GPU Memory Sharing Based Sampling

1) Hybrid Pointer-Driven Data Placement: Graph sampling demands precise access to the unevenly distributed neighbor information of target nodes, a task that becomes especially challenging across GPUs in a shared-memory setup. To address this, we devise a hybrid pointer-driven data placement strategy aimed at two crucial objectives: 1) to ensure that neighbor information is seamlessly accessible across multiple GPUs; and 2) to accurately track the head and tail addresses of each vertex's neighbor list for precise data retrieval. Moreover, we leverage the fine-grained communication features of the NVSHMEM library to facilitate efficient neighbor retrieval during the cross-GPU sampling process.

Figure 7 illustrates utilization of the Compressed Sparse Row (CSR) format for storing the graph structure, partitioning it into multiple patches. Each partition encompasses a subset of vertices along with their corresponding neighbor node IDs, residing within the memory sharing space. To facilitate efficient access to neighbor locations for the proposed memory sharing sampling, we maintain a neighbor address pointer for each GPU. This pointer records the starting and ending addresses of each vertex for memory sharing. For instance, if vertex i is stored on GPU m, the i-th entry of pointer m records the address of vertex i on GPU m, and the $(i+1)$-th entry records the address of vertex $(i+1)$ on GPU m.

With this data layout, multi-GPU sampling becomes more streamlined in the HyDRA framework. Initially, HyDRA identifies the residing GPU of a target vertex and retrieves its address using the corresponding neighbor address pointer. Data located on the local GPU is directly accessed, whereas remote data retrieval is facilitated through the NVSHMEM API function `nvshmemx_TYPENAME_get_warp`, transferring necessary data to the local shared memory. Then, vertex neighbors are processed using a warp (the smallest unit of threads that can be scheduled and executed on a streaming multiprocessor in NVIDIA GPUs), which randomly selects a set number of neighbors for sampling. Distinct from DSP's approach, the memory-sharing design in HyDRA combines sampling and communication into one kernel operation, allowing sampling warps direct access to remote neighborhood data. This method eliminates the data shuffling and cross-GPUs synchronization overhead in DSP's multi-GPU sampling strategy.

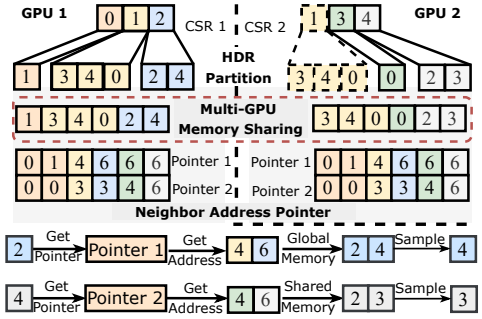


Fig. 7: Illustration of cross-GPUs sampling with hybrid pointer-driven neighbor placement and HDR partition.

2) High-degree Vertex Replication and Partitioning: Table II indicates that traditional graph partitioning methods result in significant communication overhead when paired with GPU memory sharing design, mainly due to not adequately considering high-degree vertices' impact on sampling. To address this, HyDRA implements High-Degree vertex Replication (HDR) based Partitioning, a method that stores copies of high-degree vertices in each GPU's partition to convert remote accesses into local reads, significantly reducing communication volume. As shown in Figure 7, the vertex 1 in the dotted section represents the replicated high-degree vertex.

Algorithm 1 presents our HDR Partitioning approach in detail. Initially, to significantly reduce communication overhead, we replicate the top p_r fraction of high-degree vertices across

Algorithm 1: HDR Partitioning

input : Graph $G = (V, E) = \{(v_i, v_j) \in E \| v_i, v_j \in V, where\ |V| = N\}$, number of partition M, replication rate p_r, assign batch size n
output: Partitioned graph $S = \{SG_1, \ldots, SG_M\}$

1 Calculate the number of replicated vertices $Nr = p_r * N$, and select the top N_r vertices based on their degrees to form V_r.
2 Obtain the remaining vertices $V' = \complement_V V_r$.
3 **for** $V_i \in V', where\ |V_i| = n$ **do**
4 Calculate the sum of degrees for each SG set, and select the set with the minimum sum, denoted as SG_k.
5 $SG_k = SG_k \bigcup V_i\ ,\ V' = \complement_V V_i$.
6 **end**
7 **for** $SG_j \in S, j \leftarrow 1$ **to** M **do**
8 Assign vertices to partition $SG_j = SG_j \bigcup V_r$.
9 **for** $v \in SG_j,\ find\ all\ (v, v_l) \in E$ **do**
10 Assign edges to partition $SG_j = SGj \bigcup (v, v_l)$.
11 **end**
12 **end**

all partitions (lines 1-2), a strategic step aimed at minimizing communication by efficiently converting frequent remote accesses into local accesses. Subsequently, the remainder of the vertices are distributed using a greedy algorithm to balance the communication load evenly across partitions, with vertices grouped for efficient allocation (lines 3-6). In the final phase, we ensure thorough coverage by assigning both the selectively replicated high-degree vertices and the standard edges (lines 7-12). This approach not only targets specific vertices for replication to mitigate communication peaks but also evenly distributes the rest of the graph to uphold load balance.

The HDR Partitioning strategy inherently involves a trade-off: it introduces data redundancy through the replication of high-degree vertices, thereby increasing storage demands. However, this is counterbalanced by a considerable reduction in communication overhead, particularly critical in processing real-world graphs characterized by skewed degree distributions. The effectiveness of this approach, alongside a detailed examination of the trade-off between redundancy and communication efficiency, is further explored in Section IV-D.

3) Threshold Pruning: Through leveraging HDR Partitioning, HyDRA effectively mitigates the communication surge induced by high-degree vertices. Nonetheless, residual high-degree vertices may persist, potentially hindering the efficiency of the sampling process. To further alleviate communication overhead, we propose a threshold pruning method. This method entails establishing a communication threshold θ, which is set to be greater than the number of sampled neighbors. Initially, a starting point is randomly selected from the neighbor list, transmitting a contiguous segment of neighbors equal to the threshold value, while pruning the remaining neighbors. By pruning neighbors surpassing the threshold, we can further diminish the volume of communication data. Subsequently, a number of neighbors equal to the fanout value are randomly sampled from this pruned list.

In our hybrid pointer-driven data placement approach, we

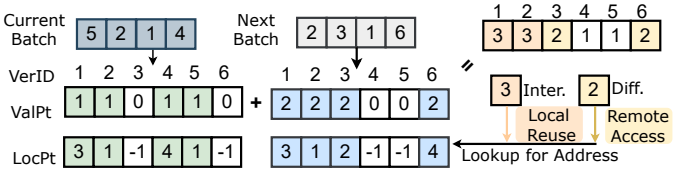


Fig. 8: Illustration of dynamic cross-batch data orchestration.

efficiently identify the pruning point to retrieve necessary data quickly. This threshold pruning technique, by limiting the amount of neighbor information transmitted for each target vertex, significantly improves the efficiency of memory-sharing sampling. A primary concern with threshold pruning, however, is its potential to affect model accuracy due to the possible exclusion of some neighbors. Nonetheless, we argue that the introduction of randomness through threshold pruning, when applying an appropriately selected threshold, can alleviate these concerns. To establish this threshold, we utilize Equation 4, where α and β are coefficients that reflect the impact of the average degree of vertices and the fanout value, respectively.

$$\text{threshold} = \alpha \cdot \text{average degree} + \beta \cdot \text{fanout} \tag{4}$$

In Section IV-E, we demonstrate that by carefully tuning the values of α and β, we can strike a balance between reducing communication overhead and preserving model accuracy.

B. Hierarchical Cross-batch Feature Retrieval

In our approach, we distribute feature data across different GPUs, which then retrieve the data needed for training via collective communication. Currently, we utilize a range-based partitioning method to divide these features. The investigation of more sophisticated partitioning schemes remains an area for future research.

1) Dynamic Cross-batch Data Orchestration: In Section II-B2, we discuss the challenges of managing the randomly interleaved distribution of Intersection and Difference segments across batches, which hinders efficient data extraction and differentiation. Traditional mapping strategies fall short in addressing this issue within GPU parallel programming environments, prompting the need for a novel approach. Our solution, a dynamic cross-batch data orchestration mechanism, is introduced to effectively manage this challenge. Illustrated in Figure 8, our mechanism, HyDRA, utilizes two specialized pointer vectors, the Value Pointer (ValPt) and the Address Pointer (LocPt), keyed to the vertex IDs of concurrent batches. These vectors, equal in length to the vertex count, leverage offset addresses to uniquely represent each vertex ID. ValPt is employed to indicate a vertex's presence in a batch, with absent vertices marked as 0, while those present are alternately labeled as 1 or 2 in successive iterations, facilitating the identification of batch-to-batch relationships.

Continuing with the dynamic cross-batch data orchestration mechanism, LocPt plays a crucial role by pinpointing the location of a vertex's feature data within batch features,

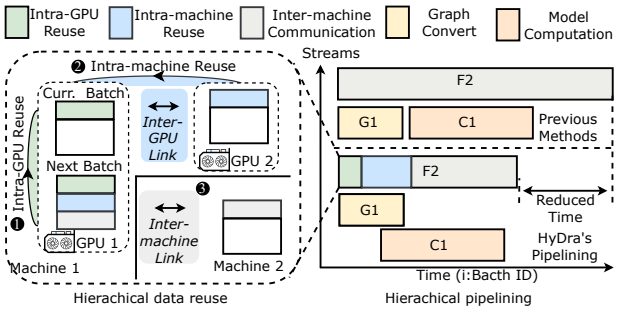


Fig. 9: Illustration of hierarchical data reuse and pipelining.

thus simplifying the process of data retrieval. Through the combined analysis of ValPts from adjacent batches—where a cumulative ValPt score of 3 signals features to be reused within the same batch, and a score of 2 highlights those requiring external fetches—we efficiently delineate between data that can be locally accessed and that which necessitates remote communication. This methodological use of ValPt and LocPt vectors across batches significantly enhances our ability to leverage GPUs' parallel processing strengths. It streamlines the identification of data overlaps between batches, thereby optimizing data handling and accelerating computation in our dynamic cross-batch data orchestration system.

2) Hierarchical Data Reuse and Pipelining: Our dynamic cross-batch data orchestration mechanism significantly minimizes communication overhead by leveraging data reuse across iterations, thereby boosting training efficiency in distributed computing environments with multiple nodes and GPUs. This approach is adapted for a multi-tiered data reuse strategy within a heterogeneous communication framework, which includes multi-GPU setups linked by NVLINK and multi-machine configurations connected via InfiniBand. As illustrated in the left part of Figure 9, HyDRA implements dynamic cross-batch data orchestration through a staged process: ❶ At the intra-GPU level, it optimizes data movement within the memory of individual GPUs. ❷ For intra-machine reuse, it treats the current batch as a composite of all batches from GPUs within a single machine, synchronizing batch IDs via all-gather communication. ❸ At the inter-machine level, it facilitates feature data exchange through all-to-all communication, ensuring efficient data retrieval across machines.

Building upon the hierarchical data reuse framework, we significantly enhance the efficiency of overlapping communication with computation. Previous efforts, like DSP [7] and Legion [44], pioneered the use of inter-batch pipeline parallelism, a strategy that involves parallelizing communication for upcoming batch feature data with computation tasks of the current batch, as depicted in Figure 9. Despite these advancements, the utility of pipeline parallelism in GNN models has been limited due to their lower computational density, where computation tasks often conclude faster than communication tasks, as detailed in Section II-B. By integrating hierarchical data reuse principles, our approach effectively minimizes communication delays using high-speed connectivity, thereby achieving a more balanced and efficient overlap between communication and computation phases in GNN models.

TABLE III: Dataset properties: vertices and edges

	ARX [48]	PRO [48]	PAP [48]	GDE [49]	MAG [50]	TWI [51]
$\|V\|$	2.9M	2.4M	111M	17K	244.2M	41.7M
$\|E\|$	30.4M	123M	1.6B	191M	1.7B	1.5B

The AGGREGATE function, critical in GNN computations as shown in Equation 1, involves integrating neighbor information—effectively multiplying a sparse adjacency matrix (symbolizing a subgraph) with a dense feature matrix [45]. Typically, this process utilizes the Compressed Sparse Column (CSC) format for the adjacency matrix in forward propagation and the Compressed Sparse Row (CSR) format in backward propagation. However, subgraphs derived from sampling are often in Coordinate (COO) format, requiring format conversion for efficient computation. Considering the requirements for communication, graph format conversion, and computation, we streamline the process through dual-level pipelining: initiating graph format conversion in tandem with feature data communication and synchronizing CSR to CSC matrix conversion with ongoing model computation. This strategy, by allowing for the concurrent execution of distinct operations, not only maximizes GPU resource efficiency but also significantly reduces the overhead associated with communication and format conversion, bolstering the overall system performance.

Our layered approach enhances data movement and utilization across various layers of communication, ensuring that HyDRA operates effectively within complex distributed computing landscapes. This approach significantly boosts the efficiency and scalability of GNN training, especially within diverse communication environments.

IV. Evaluation

HyDRA is built upon PyTorch [46] v1.13.1. We employ NVSHMEM v2.6.0 [47] to enable communication between shared GPU memory spaces, while the rest of the GPU-to-GPU communication is facilitated by NCCL v2.10.3 [36].

A. Experimental Setup

1) Physical Cluster: The experimental setup features a physical cluster consisting of 8 multi-GPU servers. Each server is equipped with dual AMD EPYC 7543 processors (2.8GHz, 32 cores). Additionally, each server houses eight Nvidia Tesla A100 40GB GPUs, resulting in a total of 64 A100 GPUs across the cluster. These servers are interconnected via a high-speed 56Gbps FDR InfiniBand network, complemented by 512GB of DDR4 2400MHz ECC memory.

2) Dataset: We conduct our experiments using multiple large-scale real-world datasets, as presented in Table III. The Arxiv dataset [48], a smaller dataset, is exclusively utilized for accuracy testing. The Products [48] and GDELT [49] datasets are on the scale of hundreds of millions edges, while the

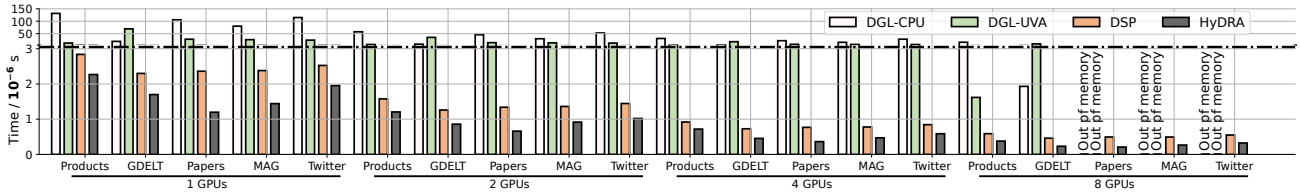


Fig. 10: Results of sampling time per vertex: performance comparison across datasets and GPU configurations.

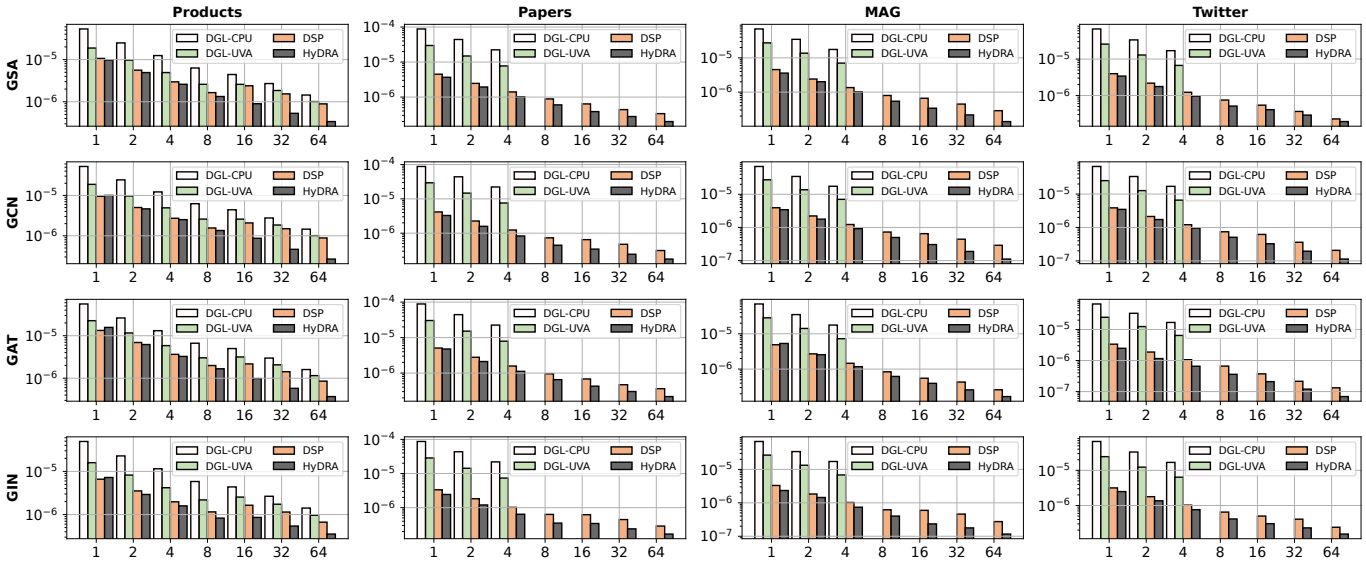


Fig. 11: Results of end-to-end training time per vertex: performance comparison across datasets and GPU configurations.

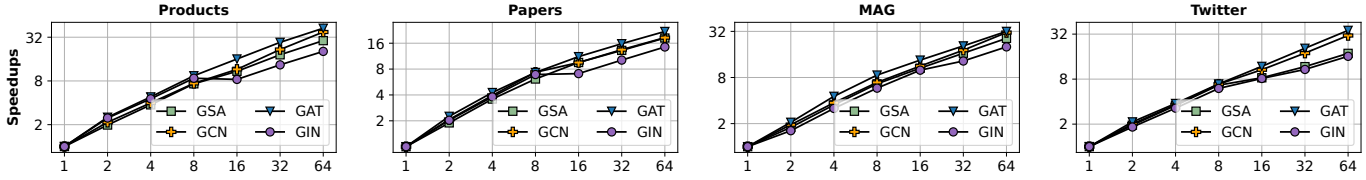


Fig. 12: Scalability analysis of HyDRA: speedup valuation on multiple datasets with varying GPU configurations.

Papers [48], MAG [50], and Twitter [51] datasets are massive, each containing over a billion graph edges.

3) Methodology: We compare HyDRA against three leading methods: DGL-CPU [15], DGL-UVA [17], and DSP [7]. DGL-CPU and DGL-UVA both manage graph data on the CPU side, with a key difference: DGL-CPU conducts sampling using CPU resources, while DGL-UVA leverages GPU sampling through UVA technology. Conversely, DSP places the graph on the GPU side and conducts sampling via all-to-all communication. Due to memory constraints on our machines, preventing the storage of feature data on the CPU, we uniformly partitioned the feature data across GPUs in a disjoint manner and utilized all-to-all communication for feature retrieval. For the network architecture, we selected the following four classic GNN models: GraphSAGE (GSA) [8], GCN [13], GAT [14] and GIN [52]. Following the recommendations on the OGB leaderboard [53], we configured the fanout settings to 10, 15, and 20 and set a default batch size of 1200, aligning our experiments with best practices for these models. Furthermore, we use the training time per vertex as the metric to evaluate training efficiency (i.e., the training time divided by the total number of vertices used for training).

B. Multi-GPUs Sampling Performance Analysis

Figure 10 illustrates the efficiency and scalability of various methods as the number of GPUs increases. Across all methods, a consistent reduction in sampling time is observed. DGL-CPU exhibits the least efficiency in sampling, with DGL-UVA following closely behind. The limited performance of these methods can be attributed to their reliance on storing graph topology on the CPU, incurring significant PCIe transfer overheads and constraining their sampling speeds. Furthermore, both DGL-CPU and DGL-UVA experience exacerbated limitations by replicating data in CPU memory for each

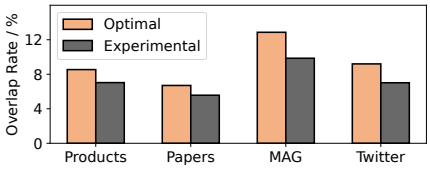


Fig. 13: Overlap rate in multi-GPU sampling kernel.

processing unit, leading to Out-of-Memory errors on billion-edge graphs due to the handling of merely topological data.

Conversely, DSP and HyDRA optimize performance by allocating graph data directly on GPUs, thereby circumventing the CPU-GPU communication bottleneck and achieving enhanced sampling efficiency. Among these, HyDRA stands out by delivering unparalleled sampling efficiency across all evaluated datasets and GPU configurations, outperforming DGL-CPU, DGL-UVA, and DSP with speedups of 28.9x, 18.9x, and 1.6x, respectively. This superior performance is primarily due to HyDRA's innovative GPU memory-sharing design and the adoption of HDR Partitioning, effectively addressing the issue of increased communication volume. Furthermore, HyDRA demonstrates exceptional scalability, achieving up to a 7.35x speedup with eight GPUs, underscoring its robust performance in multi-GPU environments.

Overlap rate. We evaluate the overlap rate between graph sampling and neighbor retrieving time in the multi-gpu sampling kernel. We denote T_l as the time required when all vertices are local, T_r as the time required when all vertices are remote, and $T_{l\&r}$ as the actual observed execution time. In addition, the proportion of local data is represented by η. Therefore, the optimal overlap rate (represented as R_o) and the experimental overlap rate (represented as R_e) are:

$$R_o = 1 - \frac{min(\eta \times T_l, (1-\eta) \times T_r)}{\eta \times T_l + (1-\eta) \times T_r} \tag{5}$$

$$R_e = 1 - \frac{T_{l\&r}}{\eta \times T_l + (1-\eta) \times T_r} \tag{6}$$

We present the optimal and experimental overlap rates in Figure 13. For all four datasets, all overlap rates are below 10%, indicating that during the sampling process, the communication overhead for neighbor access is significantly higher than the cost of local sampling. Consequently, even without mapping an interleaved execution of local and remote operations as in MGG [39], our approach achieves an overlap rate of 80% of the theoretical optimum.

C. Multi-nodes Training Scalability

Figure 11 showcases the end-to-end training times across different GPU configurations, plotting the number of GPUs on the x-axis against the training time per iteration on the y-axis. To accommodate the memory constraints of a single GPU, we reduced the feature dimensions of the Papers, MAG, and Twitter datasets to 32. Based on insights from Figure 4, we standardized the batch size to 20k for these experiments. HyDRA outperforms DGL-CPU and DGL-UVA with average speedups of 12.9x and 5.1x, respectively, highlighting its efficiency. Notably, DGL-CPU and DGL-UVA are unable to process the Papers, MAG, and Twitter datasets using more than 8 GPUs due to memory constraints. When compared to DSP, HyDRA exhibits a 1.6x improvement in training efficiency on average. Specifically, it achieves a 1.3x speedup in setups with fewer than 8 GPUs, a figure that rises to 2.0x in configurations exceeding 8 GPUs. This variance is explained by the nature of GPU interconnects: NVLink facilitates faster feature retrieval within single-node configurations, whereas cross-node communication becomes a significant hurdle in multi-node setups. HyDRA's data reuse strategy effectively addresses this challenge by reducing cross-node communication, thereby significantly boosting performance in larger configurations.

Figure 12 illustrates the scalability curves of HyDRA. Compared to the single-GPU setup, expanding the system to 64 GPUs results in a speedup ranging from 16x to 42x across different datasets. Up to 8 GPUs, the speedup shows nearly linear growth; however, beyond 8 GPUs, the rate of increase diminishes. This phenomenon arises from the shift from high-bandwidth intra-node communication to low-bandwidth inter-node communication, leading to a significant increase in communication overhead. Notably, on the Products and MAG datasets, HyDRA achieves over 30x speedup, attributable to our hierarchical data reuse and pipelining methods. These methods not only recycle redundant cross-batch data to reduce communication volume but also mitigate communication overhead by overlapping computation and communication. However, on the Papers dataset, HyDRA's performance when scaling to multiple machines is less than ideal, yielding only about a 18x speedup. As depicted in Figure 4, Papers exhibits the lowest redundancy rate, resulting in a higher volume of data necessitating additional communication. This underlines the efficacy of our hierarchical data reuse approach.

D. Sensitivity Analysis of HDR Partitioning

We explore the trade-off between sampling time and memory consumption, building on the discussions in Section III-A2. Our comparative experiments utilize the METIS partitioning method and are conducted on a 4 GPU setup. We define 'sampling time' as the average duration per iteration and 'memory increase' as the rate of memory growth for subgraph's neighbors after partitioning, relative to the baseline without data duplication. By varying the proportion of replicated high-degree vertices (p_r in Algorithm 1), we observe trends in both metrics, as shown in Figure 14, with DSP results indicated by dashed lines. Increasing the replication of high-degree vertices improves sampling rates across datasets but also leads to higher memory consumption. Remarkably, our method, without threshold pruning, reduces sampling time by 39.8% on MAG with the same memory use, and cuts memory by 93.2% on Papers while maintaining the sampling time. This highlights the effectiveness of HDR partitioning in reducing remote access to high-degree vertices with minimal memory space by leveraging redundant storage.

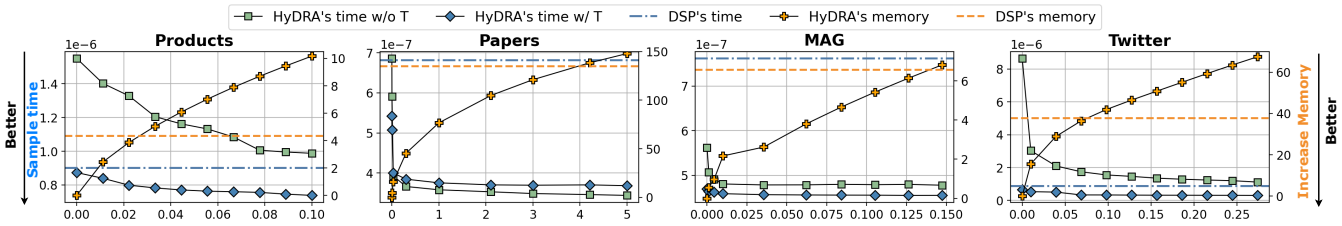


Fig. 14: Results of sampling time and memory consumption: comparison of HyDRA and DSP with varying vertex replication rate. The abbreviations "w/o t" and "w/ t" denote HyDRA without and with the threshold pruning method, respectively.

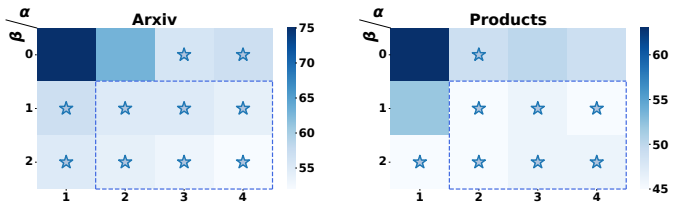


Fig. 15: Heatmap of convergence with threshold selection.

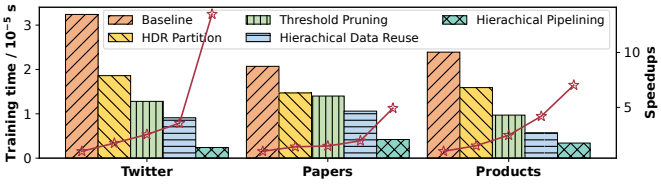


Fig. 17: Results of performance breakdown of HyDRA.

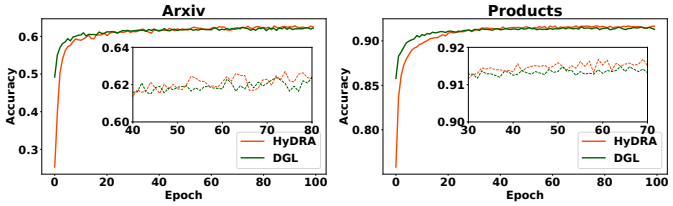


Fig. 16: Validation accuracy curve for GraphSAGE.

Importantly, the effectiveness of HDR partitioning is contingent upon the dataset's average degree. For instance, with MAG, which has an average degree of 6.92, our method surpasses DSP without needing to replicate high-degree vertices. Conversely, for datasets with significantly higher average degrees, such as Twitter (average degree of 35.9), the performance advantage diminishes. This reduction in efficiency is attributed to an increase in the number of vertices exceeding the required neighbor count for sampling, thereby elevating communication overhead, as elucidated in Equation 3. Nevertheless, we address this by implementing threshold pruning, which trims the neighbor list at a predefined threshold. This strategy effectively mitigates the communication challenge, ensuring our approach consistently outperforms DSP by minimizing communication overhead across all datasets.

In summary, our extensive experiments show that our holistic memory-sharing sampling approach markedly boosts sampling efficiency and optimizes memory overhead.

E. Accuracy Evaluation

To evaluate how parameters α and β affect model convergence, we conducted experiments using a four-GPU configuration. Figure 15 shows a heatmap for α and β values across two datasets, where lighter colors represent better model convergence observed with α and β exceeding 1. Thus, we set both α and β to 1 for standard operations, as threshold pruning within this parameter range seems to have a minimal effect on convergence. Figure 16 compares accuracy trends between HyDRA and DGL with the GraphSAGE model, showing both frameworks achieve similar convergence by the 60-epoch mark on the Arxiv dataset and by 50 epochs on the Products dataset. Notably, HyDRA maintains accuracy parity with DGL in both cases. These results highlight that careful threshold selection enables threshold pruning to maintain global neighbor visibility without compromising model accuracy or convergence.

F. Performance Breakdown Study

We conduct a performance breakdown study on HyDRA's core components with 4 GPUs. Beginning with naive shared GPU memory sampling as a baseline, we gradually integrate these components to assess performance enhancements. Figure 17 illustrates the results obtained from the Twitter, Papers and Products datasets. Upon introducing HDR partitioning, the performance of both datasets exhibits significant improvements, achieving speedup ratios of 1.74x, 1.41x and 1.51x, respectively. This highlights that HDR partitioning effectively mitigates the impact of high-degree vertices, thereby reducing communication overhead during the sampling process. Notably, datasets with higher average degrees, such as Twitter and Products, experience a more substantial improvement. Subsequently, with the addition of threshold pruning, the performance of both datasets see further enhancements of 31.1% and 4.7% and 38.9%, respectively. Threshold pruning contributes to a reduction in sampling communication volume, thereby improving sampling efficiency. Employing hierarchical data reuse to leverage redundant data between adjacent batches results in performance improvements of 28.9% , 24.2% and 41.2%, respectively. This strategy effectively reduces the communication overhead of feature retrieval. Finally, by implementing hierarchical pipelining to overlap communication

with computation and graph transformation with computation, we optimize GPU resource utilization, achieving additional speedup ratios of 3.79x, 2.52x and 1.68x respectively.

Overall, both datasets achieve impressive speedup ratios of up to 13.5x, clearly underscoring the cumulative effectiveness of each integrated component within HyDRA.

V. Related Work

A. Graph Sampling

Initial GNN systems, including AliGraph [54] and Euler [55], utilized CPU-based sampling methods. Subsequent advancements introduced by TGL [56] made use of a Temporal-CSR data structure to expedite dynamic graph operations on multi-core CPUs [57]. Despite these improvements, CPU-based sampling suffered from slow processing speeds and PCIe transfer bottlenecks between the CPU and GPU. The shift towards GPU-based sampling was marked by the introduction of frameworks such as DGL-GPU [15] and C-SAW [58], with GNNLab [20] further optimizing sample-based GNN training on GPUs. However, these approaches were limited by their reliance on single-GPU systems, which constrained scalability for processing large graphs. DGL-VUA and Quiver [59] attempted to mitigate these issues by employing VUA techniques for more efficient GPU memory use, but still faced PCIe transfer bottlenecks. DSP [7] made a significant leap by distributing the graph topology across GPUs to enable distributed sampling, which was the state-of-the-art method for multi-GPU sampling. However, the distinction between its coarse-grained neighbor retrieval and fine-grained sampling brings challenges with data shuffling and cross-GPU synchronization. Our work introduces a novel memory-sharing multi-GPU graph sampling technique, specifically designed to remove the data shuffling overhead, representing a key advancement in scalable GNN training for billion-edge graphs.

B. Feature Retrieving and Reusing

Feature caching is an established method to expedite feature retrieval, with PaGraph [18] introducing a static strategy that minimizes CPU-to-GPU transfer times by caching frequently accessed features. To tackle the high overhead associated with dynamic caches, BGL [19] implements proximity-aware reordering, enhancing the FIFO cache's efficiency. Additionally, GNNLab [20] innovates with a pre-sampling-based policy, predicting data popularity to optimize caching. Yet, these caching techniques face limitations due to the finite memory of single GPUs, struggling with cache efficiency in very large graph scenarios. HyDRA addresses these challenges by capitalizing on data overlap across batches to minimize redundant transfers. In the realm of dynamic graphs, data reuse becomes even more critical. ESDG [60] utilizes observed minimal changes in discrete dynamic graphs to lessen data transmission by identifying distinct segments. Conversely, our work deals with randomly sampled subgraphs that lack predictable patterns of change characteristic of dynamic graphs. Sven [61], [62] explores redundancy in temporal data dependencies for continuous dynamic graphs, suggesting intra-batch data reuse. However, our emphasis on inter-batch data introduces unique challenges not present in dynamically evolving graphs. Distinguishing itself, HyDRA implements a multi-level data reuse approach, innovating in the design and execution of data extraction and reuse strategies.

VI. Conclusion

In this paper, we introduced HyDRA, a pioneering framework for sampling-based GNN training on large-scale graphs. By adopting memory sharing techniques, HyDRA optimizes multi-GPU sampling and implements a hierarchical data reuse approach to boost performance in multi-node configurations. This framework combines a hybrid pointer-driven approach for topology data placement with a novel high-degree replication strategy, ensuring efficient neighbor data retrieval across GPUs and addressing the challenge of increased communication overhead. Further, through dynamic cross-batch data orchestration alongside hierarchical reuse and pipelining strategies, HyDRA capitalizes on data overlaps across batches in environments with varied communication protocols, significantly minimizing redundant data transfers. Our comprehensive evaluations reveal that HyDRA outperforms current GNN training frameworks, achieving unprecedented sampling and training acceleration on a wide array of billion-edge graphs.

VII. Acknowledgement

We thank the anonymous reviewers and our shepherd for their valuable insights and feedback. This work was supported by the National Key Research and Development Program of China (2023YFE0205700), National Natural Science Foundation of China (62302348, 62341410), Fundamental Research Funds for the Central Universities (2042023kf0132), General Program of Hubei Provincial Natural Science Foundation of China (2023AFB831), Special Fund of Hubei Luojia Laboratory (220100016) and the Science and Technology Development Fund (FDCT) Macau SAR (File no. 0078/2023/AMJ). The numerical calculations in this paper have been done on the supercomputing system in the Supercomputing Center of Wuhan University.

References

[1] Z. Wang, T. Chen, J. Ren, W. Yu, H. Cheng, and L. Lin, "Deep reasoning with knowledge graph for social relationship understanding," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 1021–1028.

[2] S. Mandal and A. Maiti, "Graph neural networks for heterogeneous trust based social recommendation," in *International joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[3] M. Cheung and J. M. Moura, "Graph neural networks for covid-19 drug discovery," in *IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 5646–5648.

[4] T. Gaudelet, B. Day, A. R. Jamasb, J. Soman, C. Regep, G. Liu, J. B. Hayter, R. Vickers, C. Roberts, J. Tang *et al.*, "Utilizing graph machine learning within drug discovery and development," *Briefings in bioinformatics*, vol. 22, no. 6, p. bbab159, 2021.

[5] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017, pp. 5115–5124.

[6] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in neural information processing systems (NeurIPS)*, 2018, pp. 5171–5181.

[7] Z. Cai, Q. Zhou, X. Yan, D. Zheng, X. Song, C. Zheng, J. Cheng, and G. Karypis, "Dsp: Efficient gnn training with multiple gpus," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2023, pp. 392–404.

[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems (NeurIPS)*, 2017.

[9] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu, "Layer-dependent importance sampling for training deep and large graph convolutional networks," in *Advances in neural information processing systems (NeurIPS)*, 2019.

[10] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining (KDD)*, 2019, pp. 257–266.

[11] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *International Conference on Learning Representations (ICLR)*, 2019.

[12] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, "Decoupling the depth and scope of graph neural networks," in *Advances in neural information processing systems (NeurIPS)*, 2021, pp. 19 665–19 679.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2016.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[15] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR workshop on representation learning on graphs and manifolds*, 2019.

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

[17] dmlc, "Deep graph library," https://github.com/dmlc/dgl/tree/0.8.x, 2022.

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

[19] T. Liu, Y. Chen, D. Li, C. Wu, Y. Zhu, J. He, Y. Peng, H. Chen, H. Chen, and C. Guo, "Bgl: Gpu-efficient gnn training by optimizing graph data i/o and preprocessing," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023, pp. 103–118.

[20] J. Yang, D. Tang, X. Song, L. Wang, Q. Yin, R. Chen, W. Yu, and J. Zhou, "Gnnlab: a factored system for sample-based gnn training over gpus," in *Proceedings of the Seventeenth European Conference on Computer Systems (EuroSys)*, 2022, pp. 417–434.

[21] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: differentiated graph computation and partitioning on skewed graphs," in *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, 2015, pp. 1–15.

[22] C. Xie, L. Yan, W.-J. Li, and Z. Zhang, "Distributed power-law graph computing: Theoretical and empirical analysis," *Advances in neural information processing systems (NeurIPS)*, vol. 27, 2014.

[23] S. Gandhi and A. P. Iyer, "P3: Distributed deep graph learning at scale," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021, pp. 551–568.

[24] M. Serafini and H. Guan, "Scalable graph neural network training: The case for sampling," *ACM SIGOPS Operating Systems Review*, vol. 55, no. 1, pp. 68–76, 2021.

[25] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of 19th International Conference on Computational Statistics (COMPSTAT)*. Springer, 2010, pp. 177–186.

[26] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations (ICLR)*, 2018.

[27] Z. Zhang, D. Yang, Y. Xia, L. Ding, D. Tao, X. Zhou, and D. Cheng, "Mpipemoe: Memory efficient moe for pre-trained models with adaptive pipeline parallelism," in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, 2023, pp. 167–177.

[28] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," in *Advances in neural information processing systems (NeurIPS)*, 2018.

[29] J. Dong, D. Zheng, L. F. Yang, and G. Karypis, "Global neighbor sampling for mixed cpu-gpu training on giant graphs," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2021, pp. 289–299.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems (NeurIPS)*, vol. 25, 2012.

[32] S. Wang, O. J. Gonzalez, X. Zhou, T. Williams, B. D. Friedman, M. Havemann, and T. Woo, "An efficient and non-intrusive gpu scheduling framework for deep learning training systems," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–13.

[33] H. Wang, D. Yang, Y. Xia, Z. Zhang, Q. Wang, J. Fan, X. Zhou, and D. Cheng, "Raptor-t: A fused and memory-efficient sparse transformer for long and variable-length sequences," *IEEE Transactions on Computers*, 2024.

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

[35] S. Wang, A. Pi, and X. Zhou, "Elastic parameter server: Accelerating ml training with scalable resource scheduling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1128–1143, 2021.

[36] NVIDIA, "Nvidia collective communications library (nccl)," https://developer.nvidia.com/nccl, 2016.

[37] J. Shun and G. E. Blelloch, "Ligra: a lightweight graph processing framework for shared memory," in *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, 2013, pp. 135–146.

[38] P. Pan and C. Li, "Congra: Towards efficient processing of concurrent graph queries on shared-memory machines," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 217–224.

[39] Y. Wang, B. Feng, Z. Wang, T. Geng, K. Barker, A. Li, and Y. Ding, "Mgg: Accelerating graph neural networks with fine-grained intra-kernel communication-computation pipelining on multi-gpu platforms," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2023, pp. 779–795.

[40] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," *http://glaros. dtc. umn. edu/gkhome/metis/metis/download*, 1997.

[41] Y. Hu, A. Levi, I. Kumar, Y. Zhang, and M. Coates, "On batch-size selection for stochastic training for graph neural networks," *https://openreview.net/forum?id=HeEzgm-f4g1*, 2020.

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

[43] S. Yang, M. Zhang, W. Dong, and D. Li, "Betty: Enabling large-scale gnn training with batch-level graph partitioning," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023, pp. 103–117.

[44] J. Sun, L. Su, Z. Shi, W. Shen, Z. Wang, L. Wang, J. Zhang, Y. Li, W. Yu, J. Zhou *et al.*, "Legion: Automatically pushing the envelope of multi-gpu system for billion-scale gnn training," in *USENIX Annual Technical Conference (ATC)*, 2023, pp. 165–179.

[45] G. Huang, G. Dai, Y. Wang, and H. Yang, "Ge-spmm: general-purpose sparse matrix-matrix multiplication on gpus for graph neural networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020, pp. 1–12.

[46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: an imperative style, high-performance deep learning library," in *Advances in neural information processing systems (NeurIPS)*, 2019, pp. 8026–8037.

[47] Nvidia, "Nvshmem communication library," https://developer.nvidia.com/nvshmem, 2022.

[48] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," in *Advances in neural information processing systems (NeurIPS)*, 2020, pp. 22 118–22 133.

[49] K. Leetaru and P. A. Schrodt, "Gdelt: Global data on events, location, and tone, 1979–2012," in *ISA annual convention*, vol. 2, no. 4. Citeseer, 2013, pp. 1–49.

[50] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "Ogb-lsc: A large-scale challenge for machine learning on graphs," *arXiv preprint arXiv:2103.09430*, 2021.

[51] J. Yang and J. Leskovec, "Patterns of temporal variation in online media," in *Proceedings of the fourth ACM international conference on Web search and data mining (WSDM)*, 2011, pp. 177–186.

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

[53] O. Team, "Open graph benchmark," https://ogb.stanford.edu/, 2020.

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

[55] Alibaba, "Euler," https://github.com/alibaba/euler, 2020.

[56] H. Zhou, D. Zheng, I. Nisa, V. Ioannidis, X. Song, and G. Karypis, "Tgl: a general framework for temporal gnn training on billion-scale graphs," in *Proceedings of the VLDB Endowment (VLDB)*, vol. 15, no. 8. VLDB Endowment, 2022, pp. 1572–1580.

[57] O. A. R. Board, "Openmp," https://www.openmp.org/, 1997.

[58] S. Pandey, L. Li, A. Hoisie, X. S. Li, and H. Liu, "C-saw: a framework for graph sampling and random walk on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis(SC)*, 2020, pp. 1–15.

[59] Quiver-team, "Quiver," https://github.com/quiver-team/torch-quiver, 2021.

[60] V. T. Chakaravarthy, S. S. Pandian, S. Raje, Y. Sabharwal, T. Suzumura, and S. Ubaru, "Efficient scaling of dynamic graph neural networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021, pp. 1–15.

[61] Y. Xia, Z. Zhang, H. Wang, D. Yang, X. Zhou, and D. Cheng, "Redundancy-free high-performance dynamic gnn training with hierarchical pipeline parallelism," in *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2023, pp. 17–30.

[62] Y. Xia, Z. Zhang, D. Yang, C. Hu, X. Zhou, H. Chen, Q. Sang, and D. Cheng, "Redundancy-free and load-balanced tgnn training with hierarchical pipeline parallelism," *IEEE Transactions on Parallel and Distributed Systems*, 2024.