Everything You Wanted to Know About Graph Neural Network Partitioning (But Were Afraid to Ask)

Chongyang Xu
cxu@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Saarland, Germany

Laurent Bindschaedler
bindsch@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Saarland, Germany

Abstract

Graph Neural Networks (GNNs) are the de facto models for deep learning on graph datasets, but training GNNs on large-scale datasets remains a challenge. Data partitioning plays a critical role in distributed mini-batch GNN training, as it directly impacts memory usage, training speed, and model accuracy. This survey comprehensively compares prevalent partitioning strategies in the Deep Graph Library (DGL) framework across standard benchmarks and models of varying depths. We systematically analyze aspects such as partition sizes, training times, memory overhead, and the accuracy associated with each partitioning method. Through this analysis, we uncover practical insights and the inherent trade-offs of these strategies. Our findings reveal surprising cases where simpler partitioning approaches outperform more sophisticated schemes. We conclude by offering practical guidelines for GNN partitioning.

CCS Concepts

• **Computing methodologies → Neural networks**; **Distributed algorithms**; **Parallel computing methodologies**; • **Information systems → Data analytics**; • **Theory of computation →** *Graph algorithms analysis.*

Keywords

Graph Neural Networks; Graph Partitioning; Distributed Training; Scalability; Load Balancing; Memory Efficiency; Empirical Evaluation; Benchmarking

ACM Reference Format:
Chongyang Xu and Laurent Bindschaedler. 2025. Everything You Wanted to Know About Graph Neural Network Partitioning (But Were Afraid to Ask). In *8th Joint Workshop on Graph Data Management Experiences Systems (GRADES) and Network Data Analytics (NDA) (GRADES-NDA '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3735546.3735857

1 Introduction

Graph Neural Network (GNN) partitioning has emerged as an essential technique for efficiently training large-scale graph datasets. GNNs are the predominant models used for machine learning tasks on graph-structured data, such as social networks [10, 13], recommendation systems [9, 18], and biological interactions [3, 5]. As real-world graphs increasingly scale to billions of nodes and trillions of edges, their storage and computational requirements often exceed the memory capacity of modern GPUs and high-end commercial machines. Therefore, effective partitioning is not just beneficial but critical for managing memory constraints, ensuring computational efficiency, and minimizing communication overhead in distributed training scenarios.

Traditionally, graph partitioning strategies, such as edge-cut and vertex-cut methods, have been used to reduce communication by minimizing the number of edges or vertices that span across partitions. Edge-cut methods focus on partitioning nodes, distributing them across partitions, while vertex-cut methods split edges, potentially replicating nodes across partitions. While effective for classical graph processing, these approaches face challenges in GNNs due to the inherent multi-hop neighborhood aggregation and the need to synchronize topology and high-dimensional feature data, leading to significant overhead. State-of-the-art techniques, such as Metis [7] and vertex-cut heuristics, aim to balance workload and reduce communication across partitions but often struggle with trade-offs in memory usage and accuracy.

This survey systematically reviews existing partitioning strategies specifically tailored to mini-batch GNN training, providing empirical insights into their practical effectiveness. Rather than introducing a single novel partitioning approach, our contribution thoroughly analyzes and compares various established methods to highlight their relative strengths and weaknesses. By consolidating empirical evidence across diverse configurations, we demonstrate that there is no one-size-fits-all solution. Instead, the effectiveness of a partitioning strategy depends on specific trade-offs between key performance metrics such as partition size, replication factor, load balance, epoch time, memory usage, and accuracy. This analysis enables us to distill generalized lessons and identify best practices tailored to different scenarios.

To rigorously evaluate and support our comparative analysis, we utilize widely recognized benchmark datasets, namely OGBN-Products [1] and OGBN-Papers [2], tested across a spectrum of partitioning configurations. We benchmark multiple representative GNN architectures, specifically Graph Convolutional Networks (GCN) [8] and GraphSAGE [4], with varying depths. Our empirical evaluations measure critical performance indicators, including partition time, average epoch time, memory usage, and accuracy metrics. By conducting these evaluations within the widely adopted DGL framework [14] and systematically comparing existing partitioning methods such as Metis, random partitioning, vertex-cut variants, and others, we provide comprehensive, reproducible evidence for our findings.

Our results reveal several notable trends and insights. Random partitioning, despite its simplicity, often outperforms advanced methods such as Metis in terms of accuracy and load balancing. However, advanced methods can be beneficial when preserving connectivity is critical. In addition, creating an excessive number of partitioning leads to significant accuracy degradation due to fragmentation, and 1-hop halo expansions introduce substantial memory overhead with little to no benefit. Finally, balancing load distribution and minimizing communication overhead remain key challenges, with simpler methods often providing more consistent performance. Overall, these findings emphasize that there is no one-size-fits-all solution, and partitioning strategies must be carefully tailored to the specific model, dataset, and resource constraints.

For instance, simple random partitioning methods often outperform advanced techniques, due to better load balancing and reduced memory overhead. Specifically, random partitioning demonstrates comparable or superior accuracy to sophisticated methods, particularly in scenarios with a moderate number of partitions. Moreover, we observe near-linear speedups in training performance as partition numbers increase to an optimal threshold (typically between 16 and 64), beyond which diminishing returns and even performance degradation may occur. Finally, deeper GNN architectures benefit more significantly from effective partitioning, as their larger receptive fields amplify communication overhead, which partitioning helps mitigate.

This paper makes the following key contributions:

- A comprehensive survey and empirical comparison of GNN partitioning strategies for mini-batch distributed training,
- The identification of several surprising insights into the interplay between partitioning complexity, memory overhead, training speed, and accuracy.
- Empirical guidelines and recommendations for selecting effective partitioning strategies.

The remainder of this paper is organized as follows: Section 2 begins by introducing some background concepts of GNNs and partitioning. Building upon this, Section 3 delves into specific partitioning methods that are currently employed. Next, Section 4 describes our experimental setup, which sets the stage for the empirical results and observations presented in Section 5. Subsequently, Section 6 highlights several surprising findings drawn from our analysis, leading into Section 7, which provides practical guidelines and actionable recommendations for practitioners. Finally, Section 8 concludes the survey by summarizing key insights and outlining promising directions for future research.

2 Background

2.1 GNN Fundamentals

Graph Neural Networks (GNNs) are neural network architectures designed specifically to process graph-structured data, integrating node features and graph topology through message-passing mechanisms. In message passing, each node iteratively aggregates feature information from its neighbors, transforming and updating its representation across multiple neural layers. This aggregation allows GNNs to capture increasingly broader context, as nodes incorporate features from multi-hop neighborhoods. Thus, multi-layer networks—such as 2-layer, 3-layer, and 4-layer GNNs—progressively extend the receptive field, exponentially increasing the number of neighbors accessed at each additional layer. In contrast to classical graph analytics, which typically involve deterministic traversals with minimal or no features, GNN workloads incorporate complex, high-dimensional embeddings, significantly amplifying computational complexity and memory demands.

2.2 Partitioning Basics

Graph partitioning strategies for distributed GNN training are primarily categorized into two types: edge-cut and vertex-cut. Edge-cut partitioning assigns nodes to different partitions, cutting edges whose endpoints reside in distinct partitions. Vertex-cut partitioning, by contrast, assigns edges to partitions, potentially replicating nodes whose edges span multiple partitions. Partitioning strategies introduce key concepts like halo nodes, replication factor, and load balancing. Halo nodes refer to nodes connected to a partition but residing externally, often included by extending partition boundaries by one or more hops. Such boundary extensions increase the replication factor, measuring data redundancy across partitions. Unequal replication or uneven distribution of training nodes across partitions can create significant load imbalance, affecting computational efficiency. Due to the multi-hop message passing nature of the computation, efficient partitioning is crucial to minimize communication overhead and manage memory usage effectively in mini-batch GNN training.

2.3 Scope of This Survey

This survey exclusively addresses mini-batch GNN training with partition-based data parallelism, where subgraphs and associated node features are sampled and processed in parallel across separate partitions. Unlike full-graph training frameworks (e.g., ROC[6] or NeutronStar[15]), which handle entire graph datasets simultaneously, mini-batch approaches iteratively process smaller subsets of the graph, making them better suited for large-scale datasets. Full-graph GNN training falls beyond the scope of this survey.

3 Partitioning Strategies

In this section, we describe the primary partitioning methods used in distributed mini-batch GNN training and covered in this survey.

3.1 Edge-Cut Methods

Random (0-hop, 1-hop). Vertices are randomly assigned to partitions, with edges connecting nodes across partitions considered as edge cuts. Optionally, partitions can include one-hop halo nodes, i.e., vertices directly connected to partition boundaries.

Metis (0-hop, 1-hop). Metis [7] is a partitioning strategy that aims to minimize edge cuts while maintaining balanced partition sizes. It supports an optional one-hop halo to include boundary edges.

3.2 Vertex-Cut Methods

VCR (random). VCR randomly assigns edges to partitions. Nodes whose edges span multiple partitions are replicated, designating the partition in which a node first appears as its primary location. This approach may result in substantial replication, particularly for high-degree nodes.

VCO (oblivious). VCO assigns edges to partitions using a heuristic that favors partitions already containing one or both vertices of the edge. When multiple partitions score equally, a balance heuristic selects partitions with fewer edges [16]. This reduces vertex cuts while maintaining balanced partition sizes.

VCD (degree-based). Similar to VCO, VCD uses a degree-based heuristic to preferentially assign edges to partitions containing the lower-degree vertex. This strategy minimizes cuts for low-degree vertices, ensuring more connected subgraphs and balanced edge distribution across partitions.

VCR-g (chunk grouping). The VCR-g approach initially partitions the graph into many small random chunks using the VCR strategy. These chunks are subsequently grouped randomly into larger partitions for each training epoch, enhancing flexibility while maintaining structure.

3.3 Other Approaches

ST (spanning-tree). The ST method employs multi-source breadth-first search (BFS) to generate a random spanning tree replicated across all partitions. Edges not included in the spanning tree are randomly assigned. ST maintains intra-partition connectivity while controlling partition sizes.

NS (neighbor sampling). NS begins with randomly selected source nodes, which are assigned to partitions, then explores multi-hop neighborhoods via random sampling with progressively decreasing sampling ratios across hops [17]. Unlike locality-optimizing approaches, NS depends solely on random sampling.

BGL (BFS-like). BGL [11] first uses BFS to generate initial graph partitions from randomly chosen nodes, stopping expansion upon reaching a maximum block size or lack of neighbors. Subsequently, small blocks are merged through a multi-level coarsening phase. A greedy heuristic assigns these merged blocks to partitions based on connectivity, size, and training nodes. The final partitions are produced through an uncoarsening step. A related community-based partitioning method is discussed in [12].

4 Experimental Setup

Datasets. We use two standard benchmark datasets for our experiments: OGBN-Products [1] and OGBN-Papers [2], whose characteristics are shown in Table 1 and are representative of typical GNN datasets. Experiments are carried out using multiple partition configurations: specifically, 1, 4, 16, 64, 256, and 1024 partitions.

Dataset	Vertices	Edges	Features	Total Size
OGBN Products [1]	2.34M	58.99M	100	4.4 GB
OGBN Papers100M [2]	105.92M	1.51B	128	103.3 GB

Table 1: Datasets used in the experiments.

GNN Models and Depths. Our evaluation includes two widely-used GNN model architectures, GraphSAGE and Graph Convolutional Networks (GCN), with network depths of 2, 3, and 4 layers. We designate the corresponding model architecture scheme and depth as `<model>-<depth>`. For example, GraphSAGE-2 corresponds to GraphSAGE with 2 layers.

Partitioning Schemes. All experiments are performed using the widely-used Deep Graph Library [14] (DGL). We utilize DGL's built-in implementations for Random and Metis partitioning. We implement and integrate custom methods such as VCD, VCR-g, ST, and NS directly into DGL. In addition, we reimplement VCR, VCO, and BGL methods within DGL to obtain consistent and comparable results.

Configurations. Experiments include configurations for both 0-hop and 1-hop partitioning. In the 0-hop setting, edges that cross partition boundaries are excluded from partitions. In contrast, the 1-hop setting includes these edges, thereby adding halo nodes to each partition. We designate the corresponding partitioning scheme and number of hops as `<scheme>-<#hops>`. For example, Random-0 corresponds to random partitioning with 0 hops (i.e., no halo nodes) whereas Metis-1 corresponds to Metis partitioning with 1 hop. Note that our ST, NS, VCR-g, and BGL implementation do not support halo nodes.

Metrics. We measure several performance indicators to evaluate the partitioning methods:

- **Partition size**: Amount of data per partition required to partition the dataset.
- **Partition time**: Time required to partition the dataset.
- **Average epoch time**: Time to complete one epoch of training across all partitions.
- **Accuracy metrics**: Training accuracy, validation accuracy, and test accuracy to assess model performance.

Hardware Configuration. We execute all experiments on a cluster of 8 identical machines. Each machine is equipped with 2 Intel CPUs (Xeon Gold 6134M), 764 GB of DDR4 memory, and two Nvidia Tesla V100 PCIe 32 GB GPUs. These machines are connected by 2× 10 GiB network (joined as one bonding interface).

5 Evaluation and Observations

In this section, we systematically evaluate various partitioning methods discussed previously. We first analyze how partitioning strategies affect memory usage and resource utilization. Next, we examine how mini-batch GNN training scales with different partitions. Subsequently, we evaluate the impact of different partitioning methods on model accuracy. Finally, we explore the trade-off between load balancing and communication overhead in partitioning.

5.1 Partition Sizes and Resource Utilization

Partitioning plays a critical role in determining the efficiency of mini-batch GNN training by directly impacting memory usage, computational load balancing, and communication overhead. Smaller partitions reduce memory constraints, balanced partitions enable efficient parallel computation, and controlled replication minimizes communication costs. However, achieving these goals simultaneously is challenging, as strategies may prioritize one aspect at the expense of others. This section empirically evaluates how different partitioning methods and configurations affect partition sizes and resource utilization, providing insights into the trade-offs involved.

Results Overview. Figures 1 and 2 show the evolution of the mean partition size with an increasing number of partitions for OGBN-Products and OGBN-Papers, respectively, under various partitioning methods and halo-hop configurations. For transparency, we also provide the exact partition sizes for OGBN-Products in

Table 2. We omit these results for the other dataset due to space constraints. Finally, we show the partitioning time taken by our implementation of the different strategies in Table 3.

How effectively do different partitioning strategies reduce partition sizes? Generally, partition sizes shrink effectively as the number of partitions grows. However, several partitioning strategies yield significantly larger partition sizes, up to three orders of magnitude more than the smallest strategy. In particular, vertex-cut partitioning results in large partition sizes due to the replication of vertices and associated features across nodes. In general, strategies that involve a higher replication of nodes to improve locality during training, including ST and NS, also suffer from larger partitions.

What is the memory overhead of expanding partition boundaries (1-hop expansions)? We observe substantial increases in partition size and memory usage with 1-hop expansions across all partitioning methods. This overhead is particularly severe for vertex-cut methods (e.g., VCR-1, VCO-1, VCD-1) and approaches such as NS, ST, and BGL, and makes it impossible to partition larger datasets such as OGBN-Papers on a single machine as it runs out of memory (shown as negative bars in Figure 1), even though the complete dataset fits comfortably in main memory. The primary source of memory overhead is the replication of node features required to maintain boundary connections, highlighting a clear trade-off between improved local connectivity and increased resource demands.

How long do partitioning strategies take? Partitioning times decrease as the number of partitions increases, but the efficiency varies significantly across methods. Random-0 consistently achieves the fastest times, with minimal variance, while methods like Metis and Random-1 take longer, particularly at lower partition counts. Vertex-cut methods (e.g., VCO-1, VCD-1) exhibit the highest partitioning times and remain slower even as the number of partitions grows. Some partitioning strategies crash at high partition counts due to memory requirements. While partitioning time may seem less critical compared to training time, it is important to note that the most expensive methods can take as much time as hundreds of training epochs, making it essential to keep this overhead in mind when selecting a partitioning strategy.

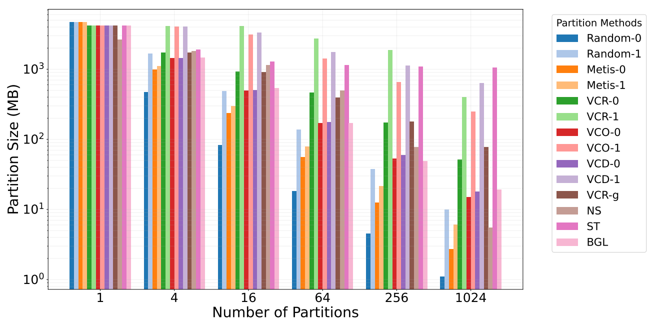

Figure 1: Partition size (MB) per number of partitions for the OGBN Products dataset. The absolute values are in Table 2.

	Number of Partitions					
Method	1	4	16	64	256	1024
Random-0	4716.8	475.2	82.7	18.4	4.5	1.1
Random-1	4716.8	1675.6	484.0	136.9	37.7	9.9
Metis-0	4716.8	989.7	238.6	55.4	12.5	2.7
Metis-1	4716.8	1112.3	295.9	79.3	21.7	6.1
VCR-0	4222.0	1737.3	924.8	463.1	172.7	51.3
VCR-1	4222.0	4153.7	4101.7	2732.7	1874.7	399.3
VCO-0	4222.0	1433.6	497.6	169.3	53.0	15.0
VCO-1	4222.0	4026.8	3142.4	1410.2	650.3	249.4
VCD-0	4222.0	1435.9	501.4	175.8	59.8	18.0
VCD-1	4222.0	4030.0	3306.2	1758.7	1126.5	633.2
VCR-g	4222.0	1733.4	908.7	393.7	178.9	77.8
NS	2632.0	1817.1	1141.6	494.3	77.5	5.5
ST	4222.0	1890.1	1293.3	1139.3	1096.8	1058.8
BGL	4222.0	1458.1	534.2	171.4	49.1	19.3

Table 2: Partition sizes in MB for different number of parti-

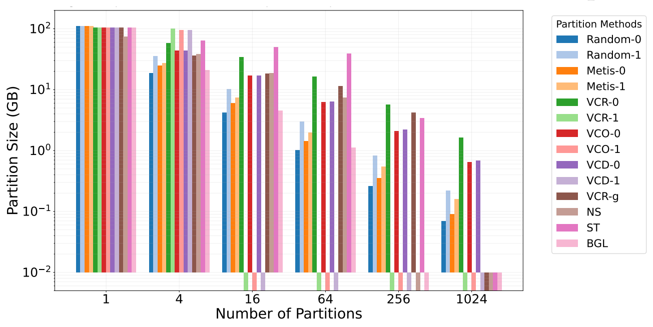

Figure 2: Partition size (MB) per number of partitions for the OGBN Papers dataset. Negative values indicate an out-of-memory error.

5.2 Impact of Partitioning on Scalability

Partitioning directly influences the scalability of GNN training by balancing computation across partitions and minimizing communication overhead. Increasing the number of partitions reduces computation time by shrinking partition sizes, but this benefit comes with trade-offs, particularly for deeper GNN models. Deeper models require aggregating information from more distant neighbors, which amplifies communication demands and increases sensitivity to partitioning strategies. This section evaluates how partitioning methods and configurations impact scalability, focusing on computation time, communication overhead, and the challenges introduced by deeper GNN architectures.

Results Overview. Figure 3 shows the mean epoch time for training GraphSAGE and GCN models with 2, 3, and 4 layers using the OGBN-Products dataset across different partitioning strategies and with an increasing number of partitions.

How well does GNN training scale based on partitioning methods and number of partitions? Epoch time generally decreases linearly with the number of partitions for most partitioning methods, which is consistent with our intuition that the epoch time is mostly driven

	Number of Partitions				
Method	4	16	64	256	1024
Random-0	475.2	82.7	18.4	4.5	1.1
Random-1	1675.6	484.0	136.9	37.7	9.9
Metis-0	989.7	238.6	55.4	12.5	2.7
Metis-1	1112.3	295.9	79.3	21.7	6.1
VCR-0	1737.3	924.8	463.1	172.7	51.3
VCR-1	4153.7	4101.7	2732.7	1874.7	–
VCO-0	1433.6	497.6	169.3	53.0	15.0
VCO-1	4026.8	3142.4	1410.2	650.3	249.4
VCD-0	1435.9	501.4	175.8	59.8	18.0
VCD-1	4030.0	3306.2	1758.7	1126.5	633.2
VCR-g	1733.4	908.7	393.7	–	–
NS	1817.1	1141.6	494.3	77.5	5.5
ST	1890.1	1293.3	1139.3	1096.8	1058.8
BGL	1458.1	534.2	171.4	49.1	–

Table 3: Partitioning time in seconds for different number of partitions on the OGBN-Products dataset. Missing values correspond to timeouts or crashes.

by the partition sizes. Vertex cut strategies, NS, and ST start to show diminishing returns beyond 16 and 64 partitions. Also, some partitioning methods, such as Random, Metis, and VCR-g tend to see performance degradation with higher number of partitions (256 and 1024). Finally, BGL uses too much memory at high partition counts and fails to complete. These slowdowns are largely due to the increased batch processing and synchronization overheads of training using more partitions and highlight the existence of an optimal partition threshold for different techniques.

How does partitioning affect different GNN models? Partitioning affects GraphSAGE and GCN models differently due to their distinct computational and communication patterns. GraphSAGE generally achieves lower epoch times compared to GCN across all partitioning methods and partition counts. This difference arises because GraphSAGE samples a fixed number of neighbors at each layer, limiting the growth of the computational graph and reducing communication overhead. In contrast, GCN aggregates features from all neighbors, leading to larger computational graphs and higher sensitivity to partition boundaries. For example, at 16 partitions, GraphSAGE consistently outperforms GCN across methods like Random, Metis, and VCR-0, with epoch times that are 30-40% lower. Despite this general difference, the scalability profiles of the two models are largely similar, with both showing diminishing returns beyond 64 partitions. However, at higher partition counts (e.g., 256 and 1024), GCN experiences slightly more pronounced slowdowns with vertex-cut methods such as VCR-1 and VCO-1 due to its higher communication demands. Therefore, the choice of partitioning strategy must be carefully aligned with the computational and communication characteristics of the model being used.

Do deeper models benefit more significantly from partitioning? In Figure 3, the duration of the training epochs increases consistently with deeper GNN models. This increase is primarily due to elevated communication requirements, as deeper layers require aggregation features from more distant nodes, which increases the sensitivity to partition boundaries. For instance, 1-hop halos in vertex-cut partitioning methods often fail to improve performance, as they cannot adequately address the need for non-local nodes with deeper models. As a result, performance differences among partitioning methods become more pronounced with increasing model depth. For example, when increasing the number of partitions from 4 to 16, the 3-layer GCN achieves a 6-fold speedup, while the 4-layer GCN sees a 10-fold speedup. These gains arise from the exponential growth in neighborhood size required by deeper layers, which partitioning effectively helps mitigate.

5.3 Impact of Partitioning on Model Accuracy

Effective partitioning not only impacts resource utilization and scalability but also influences the quality of learned representations, which affects model accuracy during training and evaluation. Intuitively, partitioning methods that preserve local connectivity and minimize fragmentation should maintain higher accuracy, while extreme partitioning or overly randomized strategies may degrade performance by limiting access to relevant neighborhood information. In this section, we empirically evaluate these effects.

Results Overview. Table 4 presents validation and test accuracy for GraphSAGE and GCN models with varying depths (2 and 3 layers) trained on the OGBN-Products dataset using different partitioning methods (Random, Metis) and partition counts (4 and 16). In addition, Figure 4 visualizes the training accuracy of the 3-layer GraphSAGE model. These results are sampled from the full set of experiments and are representative of the overall trends observed.

Does advanced partitioning improve accuracy? As shown in Table 4, the simple Random partitioning method often achieves accuracy comparable to or even better than more advanced methods, such as Metis and vertex-cut methods, at moderate partition counts (4 and 16). This trend is consistent across both GraphSAGE and GCN models, regardless of depth. This result is counterintuitive, as Random does not explicitly preserve local connectivity. However, it suggests that advanced methods, while optimizing for minimal edge cuts, may inadvertently fragment the graph in ways that disrupt the training process, such as creating partitions that are too small or unbalanced. In contrast, Random may preserve a sufficient level of connectivity across partitions by distributing nodes more uniformly, which can help maintain representation quality. These findings show that complex partitioning strategies do not always guarantee better accuracy and may even degrade performance.

How does the number of partitions affect accuracy? Figure 4 shows that the training accuracy declines sharply as the number of partitions increases to extreme levels, such as 256 and 1024. This drop is particularly pronounced for methods like Random and VCR-0, where accuracy falls by over 50% compared to lower partition counts. The decline is primarily due to excessive fragmentation, which reduces local connectivity and limits the model's ability to aggregate meaningful neighborhood information during training. In contrast, advanced methods like Metis and VCO-1 exhibit more gradual declines, suggesting that they better preserve connectivity at higher partition counts.

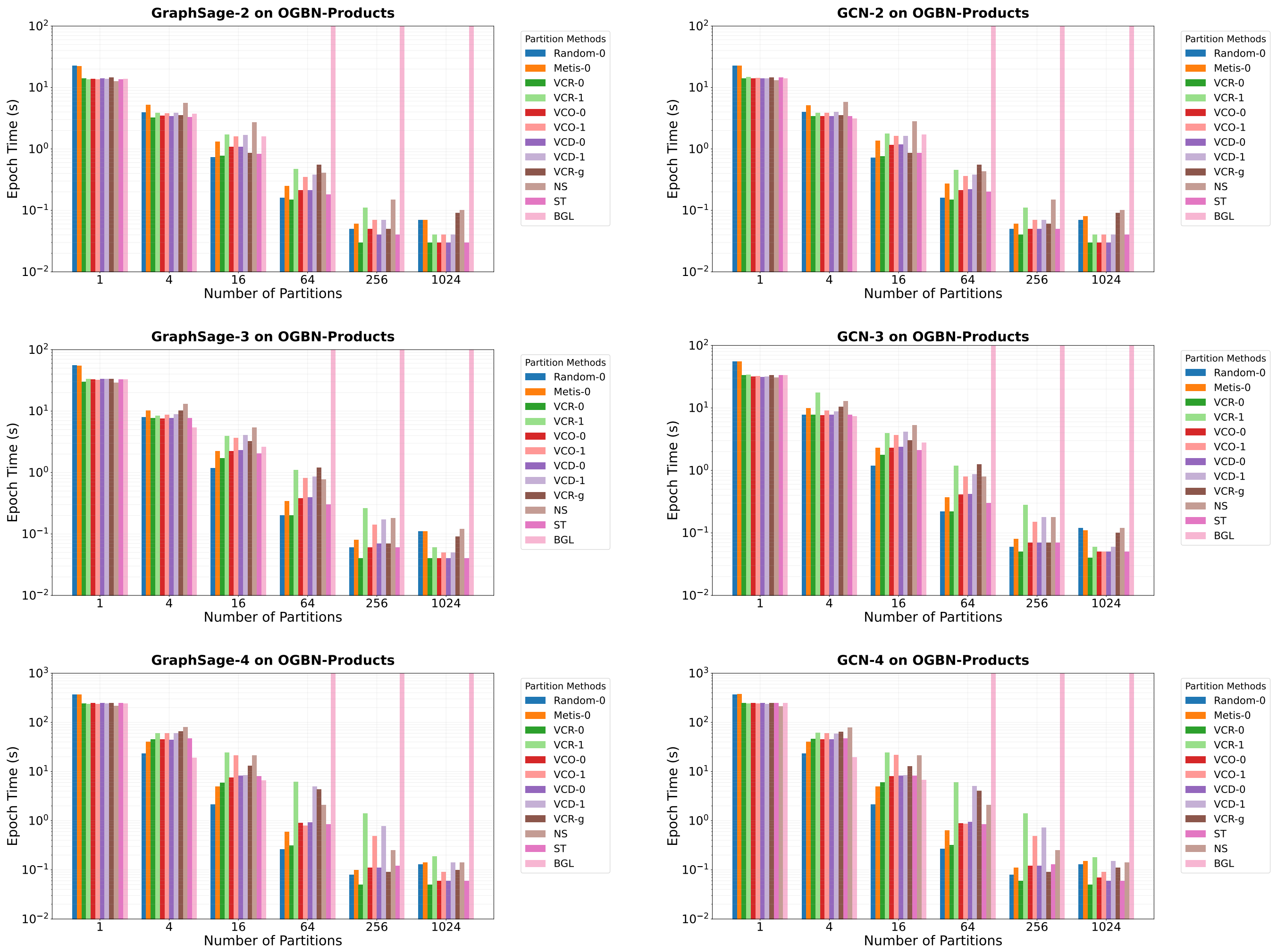

Figure 3: Epoch time comparison between GraphSAGE and GCN models on OGBN-Products with different numbers of layers.

Do certain partitioning methods perform better for specific models? The effectiveness of partitioning methods varies significantly between GraphSAGE and GCN, as shown in Table 4. For GraphSAGE, advanced methods like Metis consistently achieve high accuracy, with test accuracy reaching 0.773 for the 2-layer model at 16 partitions. In contrast, Metis performs poorly for GCN, where test accuracy drops to 0.573 under the same configuration. This disparity likely stems from the differences in how the two models aggregate neighborhood information. GraphSAGE's sampling-based aggregation is less sensitive to partition boundaries, allowing it to benefit from Metis's balanced partitions. On the other hand, GCN's full-neighborhood aggregation makes it more vulnerable to the fragmentation introduced by advanced methods, leading to better accuracy with simpler methods like Random. These results suggest that the choice of partitioning method should be tailored to the model architecture to maximize accuracy.

How does partitioning affect validation vs. test accuracy? Partitioning methods can lead to noticeable gaps between validation and test accuracy, particularly for GCN, as shown in Table 4. For example, at 16 partitions, GCN-2 achieves a validation accuracy of 0.861 with Metis but only a test accuracy of 0.573, indicating potential overfitting. This trend is less pronounced for GraphSAGE, where validation and test accuracy remain closely aligned across partitioning methods and partition counts. For instance, GraphSAGE-2 achieves a validation accuracy of 0.919 and a test accuracy of 0.773 with Metis at 16 partitions, showing better generalization. These results suggest that advanced partitioning methods like Metis and VCO may inadvertently encourage overfitting in GCN by creating partitions that optimize training performance but fail to generalize to unseen data. In contrast, GraphSAGE appears more robust to such effects, likely due to its sampling-based aggregation approach.

5.4 Partitioning Trade-offs Between Load Balancing and Communication Overhead

Partitioning methods must strike a delicate balance between evenly distributing computational loads and minimizing inter-partition communication by reducing edge cuts. Ideally, fewer edge cuts reduce communication overhead, enhancing performance, while load balancing ensures efficient utilization of computational resources.

Model	Method	4 Partitions		16 Partitions	
		Validation	Test	Validation	Test
Sage-2	Random	0.917	0.770	0.920	0.771
	Metis	0.918	0.773	0.919	0.773
Sage-3	Random	0.900	0.725	0.921	0.773
	Metis	0.892	0.735	0.919	0.777
GCN-2	Random	0.913	0.744	0.915	0.752
	Metis	0.888	0.654	0.861	0.573
GCN-3	Random	0.863	0.704	0.895	0.765
	Metis	0.887	0.696	0.893	0.690

Table 4: Validation and test accuracy for different models, depths, partitioning methods, and number of partitions.

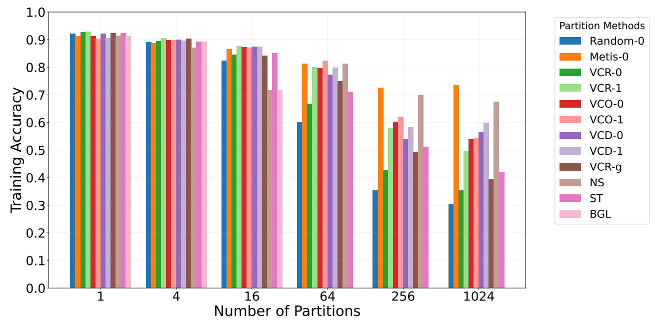

Figure 4: Train accuracy of GraphSAGE-3 on OGBN-Products. Negative bars indicate an out-of-memory error.

However, achieving this balance is challenging, as advanced methods that minimize edge cuts often introduce significant load imbalance. In this section, we explore this trade-off.

Results Overview. Table 5 compares the average partition sizes and their standard deviations for Random and Metis partitioning methods across different numbers of partitions for the OGBN-Products and OGBN-Papers datasets.

Does partitioning-induced load imbalance impact performance? As shown in Table 5, simpler methods like Random-0 often achieve better load balancing compared to advanced methods like Metis. For example, at 16 partitions on the OGBN-Products dataset, Random produces partitions with an average size of 82.66 MB and a standard deviation of only 0.38 MB, while Metis results in partitions with an average size of 238.60 MB and a much higher standard deviation of 7.30 MB. This imbalance becomes even more pronounced for larger datasets like OGBN-Papers, where Metis produces partitions with a standard deviation over 400 MB at 16 partitions. The consistent partition sizes produced by Random ensure better load distribution, which can lead to improved computational efficiency, particularly as the number of partitions increases. These results highlight that simpler methods may outperform more sophisticated strategies in scenarios where load balancing is critical.

How does load imbalance affect the benefits of minimizing edge cuts? While minimizing edge cuts generally reduces communication overhead, the resulting load imbalance can negate these benefits by introducing computational inefficiencies. For instance, Metis achieves significantly fewer edge cuts compared to Random, but its high partition size variance leads to uneven workloads across partitions. This imbalance can slow down training, as the computational bottleneck is determined by the largest partition. In extreme cases, the overhead from load imbalance can outweigh the performance gains achieved through reduced communication. These findings emphasize that minimizing edge cuts alone is insufficient for optimal performance; partitioning methods must also prioritize load balancing to fully realize their benefits.

Dataset	#Partition	Random-0	Metis-0
OGBN-Products	16	82.66 ± 0.38	238.60 ± 7.30
	64	18.40 ± 0.10	55.37 ± 4.89
	256	4.46 ± 0.04	12.48 ± 1.23
	1024	1.11 ± 0.02	2.65 ± 0.37
OGBN-Papers	16	4162.94 ± 1.97	5979.11 ± 436.14
	64	1011.15 ± 0.74	1428.69 ± 174.46
	256	250.94 ± 0.39	338.27 ± 53.14
	1024	62.62 ± 0.20	79.05 ± 21.07

Table 5: Mean partition size and standard deviation for different partition counts, strategies, and datasets.

6 Surprising Findings

Several non-obvious findings emerge from our experiments. These insights provide a deeper understanding of the interplay between partitioning strategies, memory overhead, scalability, and model accuracy. In the following, we recall, summarize, and discuss four key observations from our evaluation results.

6.1 Random Partitioning Often Outperforms Advanced Approaches

Although advanced partitioning methods (e.g., Metis and vertex-cut strategies) are designed to minimize edge cuts, our results indicate that, in many partitioning conditions, the simple Random partitioning scheme can yield superior overall performance. This outcome is largely due to Random partitioning's ability to consistently produce well-balanced partition sizes, which in turn minimizes load imbalance. In contrast, sophisticated methods may lower communication costs but increase load imbalance, leading to increased synchronization overhead that ultimately degrades performance.

6.2 1-Hop Expansions (Halo Nodes) Induce Severe Memory Overhead

An unexpected observation is the dramatic increase in memory usage when incorporating 1-hop expansions in vertex-cut methods (e.g., VCR-1, VCO-1, and VCD-1). The addition of halo nodes, which are necessary to maintain boundary connectivity, causes extensive replication of node features. This issue is worsened by the power-law degree distribution nature of many real-world graphs, causing popular vertices to be replicated in every partition. This replication often leads to out-of-memory (OOM) errors on larger datasets such as OGBN-Papers, revealing a critical trade-off between enhanced local connectivity and memory utilization. Our evaluation shows that in many cases the use of halo nodes is unjustified.

6.3 Deeper GNN Architectures Amplify Partitioning Sensitivity

Our experiments show that the benefits of partitioning become markedly more pronounced as the depth of the GNN increases. For example, while a 3-layer GCN achieves approximately a 6-fold reduction in epoch time when increasing partitions from 4 to 16, a 4-layer GCN attains roughly a 10-fold speedup. This amplified benefit is attributed to the exponential growth of the receptive field in deeper networks, which greatly increases the communication overhead. Effective partitioning mitigates this overhead, although deeper architectures are also more susceptible to memory issues under aggressive partitioning schemes.

6.4 Excessive Partitioning Leads to a Sharp Deterioration in Accuracy

While increasing the number of partitions reduces the size of each partition and enhances computational efficiency, there exists a critical threshold beyond which the accuracy of the model deteriorates sharply. For instance, at high partition counts (e.g., 1024 partitions), the fragmentation of the graph impairs the model's ability to effectively aggregate local neighborhood information, resulting in a notable drop in training, validation, and test accuracy. This observation highlights the delicate balance between computational speed and the quality of learned representations.

7 The Seven Principles of GNN Partitioning

In this section, we distill our learnings and outline seven principles that guide the effective implementation of GNN training at scale.

Principle I: Scale-Appropriate Strategy

Choose partitioning strategy based on scale:

- For small partition counts (up to 16 partitions), use simple random partitioning.
- For large-scale deployments (64+ partitions), consider vertex-cut or other approaches.
- Explore advanced methods only when simpler approaches prove insufficient.

Principle II: Memory Management

Manage memory resources efficiently:

- Do not use halos to reduce memory consumption.
- Pay special attention to high-degree node replication in vertex-cut methods.
- Let graph degree distribution inform strategy selection.

Principle III: Architectural Alignment

Match partitioning strategy to model architecture:

- For GraphSAGE, leverage advanced methods like Metis.
- For GCN, prefer simpler random approaches.
- For deep models, avoid excessive partitioning.

Principle IV: Performance-Accuracy Balance

Maintain equilibrium between performance and accuracy:

- Check advanced schemes for overfitting and imbalance risks.
- Consider random partitioning as a strong baseline.

Principle V: Full-Graph Consideration

Consider full-graph training approaches when feasible:

- Use entire graph when resources permit.
- Recognize when mini-batch training becomes necessary.
- Balance ideal approaches against practical constraints.

Principle VI: Partition Count Optimization

Optimize partition count carefully:

- Target moderate partition counts if possible.
- Monitor partition size variance.
- Account for communication overhead.

Principle VII: Load Distribution

Ensure balanced workload distribution:

- Watch for imbalance in advanced methods.
- Use simpler methods when balance is critical.
- Monitor partition size variance consistently.

8 Conclusion and Future Directions

Key Findings and Insights. This survey has highlighted several crucial insights into GNN partitioning. Simple random partitioning can often achieve comparable or superior accuracy to more complex methods, particularly for GCN models, while also offering better load balancing. Memory overhead from halo expansions, especially in vertex-cut methods, can severely limit scalability. The choice of partitioning strategy should be carefully tailored to the model architecture, dataset size, and available resources, with a focus on balancing communication overhead, load distribution, and the potential for overfitting. These findings establish a solid foundation for optimizing distributed GNN training.

Future Research Directions. Building on these findings, several challenges remain open for future research. Addressing these challenges will be critical to further advancing the scalability and efficiency of GNN partitioning strategies.

- **Improved depth-aware partitioning heuristics**: Current approaches often inadequately address the varying computational requirements across different layers of deep GNNs. Developing layer-aware partitioning strategies could substantially improve performance.
- **Dynamic repartitioning**: Switching partitioning schemes during training may prove worthwhile in many cases to maximize model accuracy while limiting the performance downsides of any one partitioning method. Unfortuantely, the potential of dynamic partition switching is limited by the lack of intelligent scheduling mechanisms. Determining optimal transition criteria while minimizing overhead presents a key research opportunity.
- **Extreme-scale memory management**: As graphs grow, novel partitioning and memory management approaches are needed to overcome memory bottlenecks while maintaining computational performance and communication efficiency.

References

[1] 2023. https://ogb.stanford.edu/docs/nodeprop/#ogbn-products.

[2] 2023. https://ogb.stanford.edu/docs/nodeprop/#ogbn-papers100M.

[3] Thomas Gaudelet, Ben Day, Arian R Jamasb, Jyothish Soman, Cristian Regep, Gertrude Liu, Jeremy BR Hayter, Richard Vickers, Charles Roberts, Jian Tang, et al. 2021. Utilizing graph machine learning within drug discovery and development. *Briefings in bioinformatics* 22, 6 (2021), bbab159.

[4] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 1024–1034. https://proceedings.neurips.cc/paper/2017/hash/5dd9db5e033da9c6fb5ba83c7a7ebea9-Abstract.html

[5] Chloe Hsu, Robert Verkuil, Jason Liu, Zeming Lin, Brian Hie, Tom Sercu, Adam Lerer, and Alexander Rives. 2022. Learning inverse folding from millions of predicted structures. In *International Conference on Machine Learning*. PMLR, 8946–8970.

[6] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, Inderjit S. Dhillon, Dimitris S. Papailiopoulos, and Vivienne Sze (Eds.). mlsys.org. https://proceedings.mlsys.org/book/300.pdf

[7] George Karypis and Vipin Kumar. 1997. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. (1997).

[8] Thomas N. Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* abs/1609.02907 (2016). arXiv:1609.02907 http://arxiv.org/abs/1609.02907

[9] Edward Elson Kosasih and Alexandra Brintrup. 2022. A machine learning approach for predicting hidden links in supply chain with graph neural networks. *International Journal of Production Research* 60, 17 (2022), 5380–5393.

[10] Srijan Kumar, William L Hamilton, Jure Leskovec, and Dan Jurafsky. 2018. Community interaction and conflict on the web. In *Proceedings of the 2018 world wide web conference*. 933–943.

[11] Tianfeng Liu, Yangrui Chen, Dan Li, Chuan Wu, Yibo Zhu, Jun He, Yanghua Peng, Hongzheng Chen, Hongzhi Chen, and Chuanxiong Guo. 2023. 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, Boston, MA, April 17-19, 2023*, Mahesh Balakrishnan and Manya Ghobadi (Eds.). USENIX Association, 103–118. https://www.usenix.org/conference/nsdi23/presentation/liu-tianfeng

[12] Zhaorui Ma, Shicheng Zhang, Na Li, Tianao Li, Xinhao Hu, Hao Feng, Qinglei Zhou, Fenlin Liu, Xiaowen Quan, Hongjian Wang, Guangwu Hu, Shubo Zhang, Yaqi Zhai, Shuaibin Chen, and Shuaiwei Zhang. 2023. GraphNEI: A GNN-based network entity identification method for IP geolocation. *Comput. Networks* 235 (2023), 109946. doi:10.1016/J.COMNET.2023.109946

[13] Xianfeng Tang, Yozen Liu, Neil Shah, Xiaolin Shi, Prasenjit Mitra, and Suhang Wang. 2020. Knowing your fate: Friendship, action and temporal explanations for user engagement prediction on social apps. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 2269–2279.

[14] Minjie Yu Wang. 2019. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*.

[15] Qiange Wang, Yanfeng Zhang, Hao Wang, Chaoyi Chen, Xiaodong Zhang, and Ge Yu. 2022. NeutronStar: Distributed GNN Training with Hybrid Dependency Management. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1301–1315. doi:10.1145/3514221.3526134

[16] Cong Xie, Ling Yan, Wu-Jun Li, and Zhihua Zhang. 2014. Distributed Power-law Graph Computing: Theoretical and Empirical Analysis. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (Eds.). 1673–1681. https://proceedings.neurips.cc/paper/2014/hash/67d16d00201083a2b118dd5128dd6f59-Abstract.html

[17] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph Edge Partitioning via Neighborhood Heuristic. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 605–614. doi:10.1145/3097983.3098033

[18] Kai Zhao, Yukun Zheng, Tao Zhuang, Xiang Li, and Xiaoyi Zeng. 2022. Joint learning of e-commerce search and recommendation with a unified graph neural network. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 1461–1469.