

GPU Kernel Optimization for Scalable Graph Neural Network Training

Lazzaro Duodo

Department of Computer Science, University of New Hampshire, Durham, NH, USA.
lazzaro.duodo@unh.edu

Ovidio Escamilla

Department of Computer Science, University of Alabama at Birmingham, Birmingham, AL, USA.
escamilla1986@uab.edu

Flavia Vigorelli

School of Computing, Clemson University, Clemson, SC, USA.
flavia.vigorelli@clemson.edu

Abstract

Graph neural networks have become a central computational method for learning over relational data, with applications ranging from recommendation and drug discovery to transportation, fraud detection, and scientific simulation. As these models scale to billions of edges and heterogeneous node types, the performance of training increasingly depends less on model expressiveness alone and more on the interaction between graph structure, memory hierarchy, parallel execution, and distributed infrastructure. This paper examines GPU kernel optimization as a system-level problem for scalable graph neural network training. It argues that kernel-level improvements cannot be understood in isolation from sampling strategy, data layout, cache behavior, communication scheduling, and governance constraints. The analysis considers irregular memory access, warp divergence, atomic contention, sparse tensor operations, and partitioning overhead as foundational bottlenecks. It then connects these bottlenecks to broader trade-offs involving throughput, latency, energy consumption, robustness, fairness, and deployment sustainability. The paper also discusses how infrastructure choices influence reproducibility, privacy, multi-tenancy, and policy compliance in production graph learning pipelines. Through cross-domain comparisons and forward-looking analysis, the paper positions GPU kernel optimization as a socio-technical design practice rather than a narrow implementation exercise. The conclusion emphasizes that scalable graph neural network training requires coordinated advances in hardware-aware algorithms, runtime systems, benchmarking standards, and governance frameworks. Such coordination is necessary for graph learning to remain efficient, equitable, and environmentally responsible as datasets and models continue to grow.

Keywords

graph neural networks; GPU kernel optimization; scalable training; systems architecture; memory hierarchy; distributed infrastructure; sustainability; fairness; robustness; governance.

1. Introduction

Graph neural networks have emerged as a powerful family of machine learning models for structured data, enabling inference and representation learning over graphs that encode social

relations, molecular interactions, transportation networks, knowledge bases, and communication systems [1]. Early formulations established the principle of recursive neighborhood aggregation, while later architectures such as graph convolutional networks and graph attention networks made graph learning practical for semi-supervised classification, link prediction, and relational reasoning [2], [3]. Inductive methods further expanded the scope of graph learning by allowing models to generalize to unseen nodes and subgraphs, which is essential in dynamic production environments [4]. Theoretical work has clarified the expressive limits of message-passing architectures and motivated architectural refinements that balance depth, width, and neighborhood sampling [5]. Surveys of the field have documented rapid methodological growth, but they also reveal that scalability remains a persistent obstacle [6], [7].

The scalability challenge is not merely a matter of adding more GPUs or increasing batch size. Graph neural network training exhibits irregular computation, sparse memory access, neighborhood explosion, and highly variable per-node workloads. These characteristics interact poorly with conventional deep learning kernels, which are optimized for dense tensor operations with regular shapes. As a result, training performance is often constrained by memory latency, cache misses, atomic contention, and inter-device communication rather than by raw floating-point throughput. Kernel optimization therefore becomes a systems problem that spans data structures, scheduling, partitioning, synchronization, and hardware utilization. It also becomes a governance problem because training efficiency influences cost, access, environmental impact, and the feasibility of auditing large models.

This paper develops a system-level analysis of GPU kernel optimization for scalable graph neural network training. It treats kernels as components within a broader computational stack that includes graph storage formats, sampling engines, feature caches, collective communication libraries, cluster schedulers, and policy controls. The discussion emphasizes structural trade-offs rather than isolated performance tricks. In particular, it examines how kernel design choices affect robustness, fairness, sustainability, and deployment viability. The central argument is that scalable graph learning requires co-design across algorithmic, architectural, and institutional layers. Optimizing a kernel without considering data governance or energy accounting may improve a benchmark score while producing fragile or inequitable systems. Conversely, governance goals such as reproducibility, privacy, and equitable access depend on concrete infrastructure decisions about memory management, partitioning, and runtime scheduling.

2. Workload Characteristics and System Implications

Graph neural network workloads differ from conventional deep learning workloads in ways that directly shape GPU kernel design. Message-passing models repeatedly aggregate features from neighboring nodes, and the number of neighbors can vary by orders of magnitude across a graph. This degree imbalance creates load skew: a small number of high-degree nodes may dominate computation, while many low-degree nodes leave execution units underutilized. Sampling techniques are commonly used to bound neighborhood size, but they introduce stochasticity and can affect convergence, fairness, and reproducibility [4], [5]. Full-batch training avoids sampling noise but may exceed device memory for large graphs, forcing partitioning and communication strategies that introduce new bottlenecks [6], [7].

The memory behavior of graph neural networks is equally challenging. Node features and embeddings are typically stored in dense arrays, while graph connectivity is stored in sparse formats such as compressed sparse row or coordinate lists. Gathering neighbor features

therefore produces irregular memory accesses that map poorly to GPU cache lines and memory coalescing rules. When many threads in a warp access non-contiguous addresses, the hardware may issue multiple memory transactions, reducing effective bandwidth. Feature caching can mitigate this problem by retaining frequently accessed node features in faster memory, but cache capacity is limited and replacement policies must account for graph topology and training phase [14]. The design of GPU kernels must therefore coordinate sparse traversal, dense feature movement, and cache residency.

The computational graph also changes across layers and epochs. In sampled training, each mini-batch induces a different subgraph, so kernel launch configurations, memory layouts, and partitioning decisions may need to adapt dynamically. In full-graph training, the topology is stable but the feature and gradient buffers are large, making memory reuse and synchronization critical. Benchmark suites such as the Open Graph Benchmark have helped standardize evaluation across datasets and tasks, but they do not fully capture production concerns such as streaming updates, heterogeneous node types, or privacy constraints [19]. Design-space studies have shown that architectural choices interact strongly with dataset properties, which means that a kernel optimized for one graph family may perform poorly on another [20].

These workload characteristics have implications beyond runtime. Irregular training can increase energy consumption because hardware waits on memory or performs redundant work. It can also complicate fairness analysis because sampling may underrepresent low-degree or minority subpopulations. If kernel optimizations privilege high-degree nodes or densely connected communities, the resulting model may inherit structural bias. System designers must therefore consider not only speed but also the distribution of computational effort across the graph. Scalable graph learning requires kernels that are efficient, but it also requires them to be predictable, auditable, and adaptable to diverse graph structures.

3. GPU Architecture and Kernel-Level Bottlenecks

GPU execution is organized around streaming multiprocessors, warps, and hierarchical memory. Dense matrix multiplication and convolution kernels exploit this architecture effectively because data can be tiled into shared memory and reused across threads. Graph kernels, by contrast, must handle sparse and irregular access patterns. Warp divergence occurs when threads in the same warp follow different control paths or access scattered memory locations, reducing instruction-level efficiency. Atomic operations used for gradient aggregation can serialize updates when many threads update the same node or edge embedding. These bottlenecks are not incidental; they arise from the mismatch between graph topology and the assumptions of GPU hardware.

Recent kernel research has explored sparsity-aware tiling, hierarchical aggregation, and adaptive workload balancing to address these issues [8]. Such approaches demonstrate that substantial speedups are possible when kernels exploit structural properties of graphs rather than treating them as generic sparse matrices. However, the gains are often sensitive to graph degree distribution, feature dimensionality, and batch construction. A kernel that performs well on citation networks may struggle on social networks or molecular graphs because the ratio of computation to memory traffic differs. This sensitivity means that kernel optimization must be evaluated across diverse workloads and integrated with autotuning or runtime adaptation.

Framework-level design also shapes kernel behavior. PyTorch Geometric and Deep Graph Library provide abstractions for message passing, batching, and neighborhood sampling, but their performance depends on underlying kernels and memory layouts [9], [10]. Platform systems such as AliGraph illustrate how storage, sampling, and training can be co-designed for industrial-scale graphs [11]. Distributed multi-GPU graph processing systems show that partitioning and communication can dominate runtime when graphs exceed device memory [12]. These systems reveal a recurring trade-off: more aggressive partitioning improves memory fit but increases communication and synchronization overhead. Kernel optimization must therefore be considered alongside partitioning and scheduling policies.

The interaction between kernels and communication is especially important in multi-GPU training. Gradient synchronization, feature exchange, and sampled subgraph transfer all consume bandwidth. Overlapping computation and communication can hide some latency, but it requires careful dependency management and may increase memory pressure. Kernel fusion can reduce launch overhead and intermediate memory traffic, but it can also reduce flexibility and complicate debugging. The optimal design often depends on the scale of the graph, the number of devices, and the latency of the interconnect. Consequently, GPU kernel optimization for graph neural networks is not a one-time implementation task; it is a continuous process of profiling, adaptation, and system-level negotiation.

4. System-Level Trade-offs in Scalable GNN Training

Scalable graph neural network training involves several recurring trade-offs. The first is between sampling and full-batch computation. Sampling reduces memory and compute per step, enabling training on large graphs, but it introduces variance and may require more epochs to converge. Full-batch training provides stable gradients but demands partitioning, caching, and communication strategies that can erase the benefits of larger batch sizes. Systems such as DistDGL address this by distributing graph partitions across machines while coordinating sampling and training [13]. PaGraph focuses on computation-aware caching to reduce feature movement and improve GPU utilization [14]. These systems show that the sampling strategy and the memory policy are inseparable from kernel performance.

The second trade-off concerns partitioning granularity. Fine-grained partitions improve load balance but increase metadata and communication overhead. Coarse partitions reduce communication but may create stragglers when a partition contains high-degree nodes or dense communities. NeuGraph explores parallel deep neural network computation on large graphs, highlighting the need to coordinate graph traversal with neural network execution [15]. GNNAdvisor proposes adaptive runtime strategies for GNN acceleration on GPUs, reflecting the importance of workload-aware scheduling [16]. P3 examines distributed deep graph learning at scale and illustrates how partitioning, caching, and communication can be jointly optimized [17]. These examples suggest that kernel optimization cannot be separated from the partitioning and scheduling layer.

The third trade-off involves memory capacity and recomputation. Storing intermediate embeddings and gradients consumes substantial memory, especially for deep models and high-dimensional features. Checkpointing and recomputation can reduce memory pressure but increase compute cost. Kernel fusion can avoid writing intermediates to global memory, but it may reduce occupancy or complicate irregular access. GNNLab proposes a factored system for sample-based GNN training over GPUs, showing that separating sampling and training can improve resource utilization [18]. Such designs indicate that system architecture often

matters more than individual kernel micro-optimizations. A fast kernel inside a poorly balanced pipeline may still yield low end-to-end throughput.

The fourth trade-off concerns latency and throughput. Production systems often require both low-latency inference and high-throughput training. Training kernels can be optimized for large batches, but inference may require small-batch or single-node updates. This divergence encourages modular designs in which training and inference share graph storage but use different execution paths. Survey work on computing graph neural networks emphasizes that algorithm, system, and hardware choices must be co-designed to avoid bottlenecks shifting from one layer to another [21]. The design space for graph neural networks further shows that many architectural choices interact, making isolated optimization unreliable [20]. Scalable training therefore depends on holistic evaluation rather than single-kernel benchmarks.

5. Infrastructure, Deployment, and Governance

Deploying scalable graph neural network training in production requires infrastructure that supports multi-tenancy, isolation, reproducibility, and compliance. GPU clusters are shared among teams and workloads, so scheduling policies influence both performance and fairness. A job that monopolizes high-bandwidth memory or interconnect bandwidth can degrade other tenants. Kernel optimization that increases resource intensity may improve one job while reducing overall cluster efficiency. Governance therefore includes resource quotas, priority classes, and observability tools that make contention visible. These mechanisms are not external to kernel design; they shape the incentives for optimizing memory footprint, communication patterns, and energy use.

Data governance is another critical dimension. Graphs often represent sensitive relationships, such as social ties, financial transactions, or medical records. Training kernels may access node features and edge lists that are subject to privacy regulations. Techniques such as differential privacy, secure aggregation, and federated learning impose additional computation and communication patterns. Kernel optimizations must respect these constraints rather than bypass them for speed. For example, caching sensitive features on a GPU may reduce latency but raise questions about data retention and isolation. Distributed training may improve scale but complicate audit trails and cross-border data transfer. Infrastructure design must therefore align with legal and ethical requirements.

Reproducibility is closely tied to infrastructure. Graph sampling, partitioning, and non-deterministic atomic operations can produce different results across runs. If kernel optimizations introduce nondeterminism or depend on hardware-specific behavior, reproducing published results becomes difficult. Benchmarks such as the Open Graph Benchmark help standardize datasets and metrics, but they do not fully specify system-level conditions such as cluster topology, interconnect, or caching policy [19]. Governance frameworks should require documentation of hardware, software, sampling seeds, and partitioning strategies. Without such transparency, claims about scalability may not generalize. Fairness also depends on infrastructure because unequal access to GPU resources can concentrate the ability to train large graph models in a few organizations [6], [7].

6. Robustness, Fairness, and Policy Implications

Robustness in graph neural networks is affected by both model architecture and system implementation. Adversarial perturbations can add or remove edges, change node features, or manipulate subgraph sampling. Kernel-level choices may influence robustness indirectly by determining which nodes are sampled, cached, or processed first. If high-degree nodes

dominate computation, adversarial actors may target those nodes to cause disproportionate disruption. If caching policies favor frequently accessed features, rare or newly added nodes may be underrepresented. Robustness therefore requires stress testing across graph perturbations and system conditions, not only clean benchmark accuracy [5], [6].

Fairness concerns arise when graph structure encodes historical inequality. Recommendations, credit scoring, and predictive policing are domains where relational data can reproduce or amplify bias. Graph neural networks can propagate features across neighborhoods, so bias in one community may spread to connected nodes. Sampling strategies can exacerbate this by either oversampling or undersampling certain communities. Kernel optimizations that improve throughput on dense subgraphs may systematically deprioritize sparse or marginalized communities. Fairness-aware training may require additional objectives, constraints, or data augmentation, all of which affect kernel workloads. System designers should provide mechanisms for monitoring per-group performance and resource allocation during training [20], [22].

Policy implications are increasingly significant. Regulations concerning automated decision-making, data protection, and critical infrastructure require documentation, human oversight, and contestability. Graph neural networks used in financial fraud detection, healthcare, or energy grids may be classified as high-risk systems. Training infrastructure must support audit logs, model versioning, and access controls. GPU kernel optimization can contribute to compliance by reducing the cost of repeated audits or privacy-preserving training, but it can also undermine compliance if it encourages undocumented approximations. Policymakers and engineers must therefore collaborate on standards for graph data provenance, model documentation, and performance reporting. Technical efficiency and democratic accountability are not separate agendas; they intersect in the design of training systems.

7. Sustainability and Resource Efficiency

The environmental cost of large-scale graph neural network training is a growing concern. GPUs consume substantial energy, and data movement between memory, storage, and network devices adds further overhead. Irregular graph kernels often waste energy because execution units stall on memory latency or perform redundant atomics. Improving kernel efficiency can therefore reduce carbon emissions, but only if the overall system is designed to avoid rebound effects. Faster training may encourage larger models and datasets, offsetting energy savings. Sustainability requires metrics that account for total energy per useful training run, not just throughput per GPU-hour [21].

Several strategies can improve sustainability. Computation-aware caching reduces redundant feature movement, which lowers memory bandwidth and energy use [14]. Adaptive runtime systems can choose kernel configurations that balance occupancy and memory traffic for the current graph [16]. Factored training systems can separate sampling from model computation, allowing each phase to run on hardware suited to its characteristics [18]. Sparsity-aware kernels can skip zero or masked computations, but they must avoid irregular overhead that negates the savings. Quantization and low-precision arithmetic can reduce memory and compute costs, though they may affect accuracy and fairness. These techniques should be evaluated holistically, because an optimization that helps one metric may harm another.

Deployment sustainability also involves hardware lifecycle and supply chains. GPU manufacturing has significant environmental and social impacts, so extending device useful life and improving utilization are important. Multi-tenant clusters with fair scheduling can

increase utilization, but they require isolation and accounting mechanisms. Federated or edge-based graph learning can reduce data movement and central energy use, but it introduces coordination overhead and heterogeneity. Sustainable graph learning is therefore a systems governance challenge. It requires transparency about energy consumption, incentives for efficient model design, and policies that prevent efficiency gains from being absorbed by unchecked growth.

8. Case Illustrations and Cross-Domain Comparisons

Recommendation systems provide a demanding case for scalable graph neural network training. User-item graphs are large, dynamic, and highly skewed. Training must handle new users, new items, and changing interactions without full retraining. Kernel optimization must therefore support incremental updates, caching, and low-latency embedding retrieval. Systems such as AliGraph illustrate the need for integrated storage, sampling, and training pipelines [11]. GNNLab demonstrates how factored resource management can improve sample-based training on GPUs [18]. In this domain, throughput and freshness are both critical, and governance concerns include privacy, manipulation, and transparency.

Drug discovery and molecular property prediction offer a different profile. Graphs are smaller but chemically rich, with heterogeneous atoms and bonds. Training may involve three-dimensional geometry and equivariant operations, which stress different kernel paths. Robustness is essential because erroneous predictions can affect health outcomes. Fairness concerns relate to representation of rare diseases and underrepresented populations. Kernel optimizations that exploit sparsity or symmetry can improve performance, but they must preserve physical and chemical validity. Benchmarks such as Open Graph Benchmark provide datasets for molecular tasks, though production deployment requires additional validation and regulatory documentation [19].

Power grids and transportation networks are safety-critical graph domains. Training on sensor networks, topology, and time-series data requires robustness to missing or corrupted inputs. Kernels must handle streaming updates and strict latency bounds. Distributed training may be constrained by data sovereignty and critical infrastructure regulations. Cross-domain comparison shows that graph neural network scaling is not a single problem. The optimal balance of sampling, caching, partitioning, and precision depends on domain-specific requirements. A kernel optimized only for benchmark accuracy may fail in operational settings where interpretability, auditability, and fault tolerance are paramount [20], [21].

9. Open Challenges and Future Directions

Several open challenges remain. First, dynamic graphs require kernels that can adapt to changing topology without expensive repartitioning. Second, heterogeneous hardware including CPUs, GPUs, accelerators, and edge devices requires portable abstractions that do not sacrifice performance. Third, privacy-preserving graph learning needs efficient kernels for encrypted or federated computation. Fourth, evaluation standards must capture system-level properties such as energy, fairness, robustness, and reproducibility, not only accuracy and runtime [8], [19], [22]. Fifth, governance mechanisms must keep pace with technical capabilities, ensuring that scalable training does not outrun accountability.

Future research should pursue hardware-software co-design that treats graph structure as a first-class input to compilation and scheduling. Kernel generators could use degree distribution, feature sparsity, and partition layout to synthesize specialized code. Runtime systems could monitor contention, energy, and fairness metrics, then adapt sampling and

batching policies accordingly. Distributed training frameworks could expose communication and memory trade-offs as tunable parameters with clear governance implications. Educational and policy efforts should also prepare engineers and regulators to reason about graph data, relational inference, and infrastructure externalities. The goal is not only faster training but also more responsible scaling.

10. Conclusion

GPU kernel optimization for scalable graph neural network training is a deeply interdisciplinary undertaking. It requires understanding of graph algorithms, GPU architecture, memory hierarchies, distributed systems, and the social contexts in which graph learning is deployed. This paper has argued that kernel-level performance is inseparable from sampling strategy, partitioning, caching, communication, energy use, robustness, fairness, and governance. The most promising advances combine sparsity-aware kernels with adaptive runtime systems and infrastructure policies that account for total system behavior. As graph datasets and models continue to grow, the field must move beyond isolated benchmark improvements toward holistic evaluation. Scalable graph learning should be efficient, but it must also be reproducible, equitable, environmentally responsible, and accountable. Achieving these goals will require sustained collaboration among computer architects, machine learning researchers, systems engineers, policymakers, and domain experts. The future of graph neural network training depends not only on faster kernels but on wiser system design.

References

1. Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1), 61–80. <https://doi.org/10.1109/TNN.2008.2005605>
2. Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*. <https://arxiv.org/abs/1609.02907>
3. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations*. <https://arxiv.org/abs/1710.10903>
4. Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems* (Vol. 30, pp. 1024–1034).
5. Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations*. <https://arxiv.org/abs/1810.00826>
6. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1), 4–24. <https://doi.org/10.1109/TNNLS.2020.2978386>
7. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1, 57–81. <https://doi.org/10.1016/j.aiopen.2021.01.001>

8. Peng, H., Xie, X., Shivdikar, K., Hasan, M. A., Zhao, J., Huang, S., ... & Ding, C. (2024, April). Maxk-gnn: Extremely fast gpu kernel design for accelerating graph neural networks training. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (pp. 683-698).
9. Fey, M., & Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*. <https://arxiv.org/abs/1903.02428>
10. Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., & Li, J. (2019). Deep Graph Library: A graph-centric, highly-performant package for graph neural networks. *arXiv*. <https://arxiv.org/abs/1909.01315>
11. Zhu, R., Zhao, K., Yang, H., Lin, W., Zhou, C., Ai, B., Li, Y., & Zhou, J. (2019). AliGraph: A comprehensive graph neural network platform. *Proceedings of the VLDB Endowment*, 12(12), 2094–2105. <https://doi.org/10.14778/3352063.3352127>
12. Jia, Z., Kwon, Y., Shipman, G., McCormick, P., Erez, M., & Aiken, A. (2019). A distributed multi-GPU system for fast graph processing. *Proceedings of the VLDB Endowment*, 12(5), 491–504. <https://doi.org/10.14778/3303753.3303758>
13. Zheng, D., Ma, C., Wang, M., Zhou, J., & Li, J. (2020). DistDGL: Distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)* (pp. 36–44). <https://doi.org/10.1109/IA351965.2020.00011>
14. Liu, Z., Zhou, K., & Li, F. (2020). PaGraph: Scaling GNN training on large graphs via computation-aware caching. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (pp. 401–415). <https://doi.org/10.1145/3419111.3421281>
15. Ma, L., Yang, Z., Miao, Y., Xue, J., Wu, M., Zhou, L., & Dai, Y. (2019). NeuGraph: Parallel deep neural network computation on large graphs. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (pp. 443–458). USENIX Association.
16. Wang, Y., Feng, B., Li, G., Li, S., Xie, Y., & Ding, Y. (2021). GNNAdvisor: An adaptive and efficient runtime system for GNN acceleration on GPUs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)* (pp. 515–531). USENIX Association.
17. Gandhi, S., & Iyer, A. P. (2021). P3: Distributed deep graph learning at scale. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)* (pp. 551–568). USENIX Association.
18. Yang, J., Tang, D., Wang, X., Wang, X., Wang, C., Chen, L., Wang, X., Wang, F., & Chen, L. (2022). GNNLab: A factored system for sample-based GNN training over GPUs. In *Proceedings of the 17th European Conference on Computer Systems (EuroSys '22)* (pp. 1–16). ACM. <https://doi.org/10.1145/3492321.3519555>
19. Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., & Leskovec, J. (2020). Open Graph Benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 22118–22133).
20. You, J., Ying, R., & Leskovec, J. (2020). Design space for graph neural networks. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 17009–17021).

21. Abadal, S., Jain, A., Guirado, R., López-Alonso, J., & Alarcón, E. (2021). Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys*, 54(9), Article 191. <https://doi.org/10.1145/3477141>
22. Dong, Y., Ma, J., Chen, C., & Li, J. (2023). Fairness in graph mining: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(10), 10583–10598. <https://doi.org/10.1109/TKDE.2022.3180108>